

誰でも
誰にでも
わかる

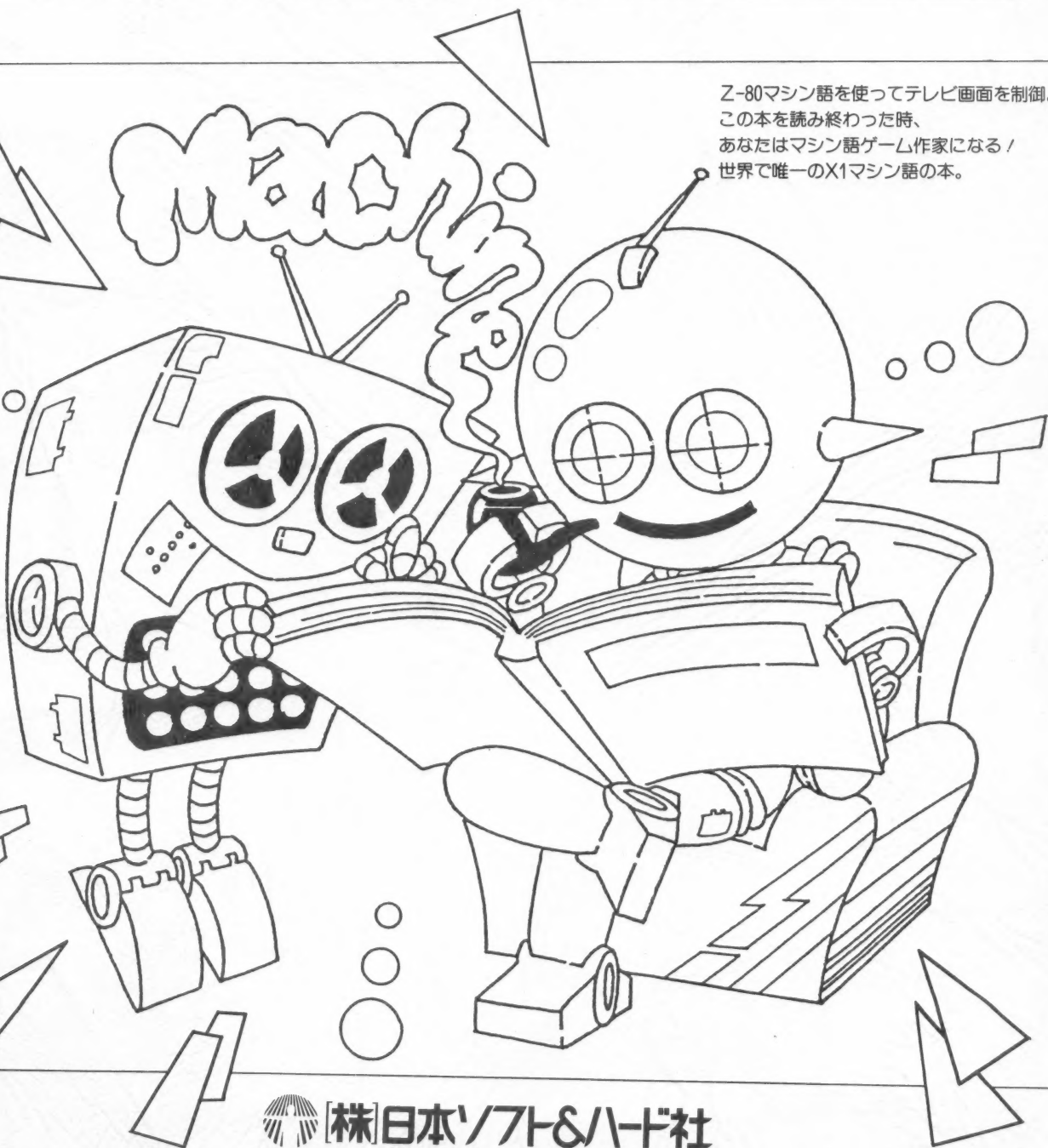


X1・X1C
X1D 対応!

清水 保弘 著

マシン語入門

Z-80マシン語を使ってテレビ画面を制御。
この本を読み終わった時、
あなたはマシン語ゲーム作家になる！
世界で唯一のX1マシン語の本。



株 日本ソフト&ハード社

ま え が き

シャープX1の全てのユーザーにこの本を贈ります。

X1ユーザーの皆さんの家の本棚には、多くのマイコン雑誌と並んで、何冊かの「パソコン入門」的な本が置かれていると思います。皆さんは、これらの中で、最後まできちんと読み通した本は何冊ありますか？ おそらく一冊もないのではないのでしょうか。今まで市販されていたパソコン関係の本の多くは、どれも記述が難かしく、初心者にとって理解に苦しむ点が多いものばかりでした。それでも向上心おうせいな皆さんは、マニュアル等を参考にして、どうにかBASICのコマンドの使い方くらいは身に付けられたでしょう。

BASICは実に親切な言語ですから、詳しい入門書などなくても、こちらが間違いをおかせばBASICがちゃんとそれを指摘してくれます。BASICが皆さんの先生がわりをしてくれたわけですね。やがて卒業の日が来ました。そしてマシン語入門……。ところが、ここで皆さんは大きな壁に突き当たることになるのです。

今まではBASICが先生がわりを務めてくれていましたが、今度は、皆さん自身が先生になってコンピューターに仕事の手順を教えてやる立場に立たされたわけなのです。

さあ大変！ 皆さんはコンピューターに仕事の仕方を正しく教えられますか？ マシン語の場合、エラー表示などというものはありませんから、皆さんが一か所でも間違えた教え方をすると、X1のZ80CPUはすぐにヘソを曲げてしまいます。ヘソを曲げたZ80は、時には皆さんの先生であるBASICに八つ当たりをして、BASICシステムをメチャメチャに壊してしまうことさえあります。これでは、さすがの皆さんでも詳しい教科書が必要になりますね。

そこで、ヘソ曲がりのZ80のなだめ方を、皆さんにだけ、こっそりお教えするのがこの本の役目です。X1のZ80CPUは、この本を読んだアナタの言うことなら、ヘソなど曲げずに素直に聞いてくれることでしょう。そして今度は、アナタと一緒にゲームをして楽しもうと提案してくるでしょう。Z80CPUは、本当は素直でいい子ばかりなのです。どうか、このカワイイZ80を末長く可愛がってあげて下さい。

この本は、マシン語初心者のマシン語初心者によるマシン語初心者のためのマシン語入門書です。初心者にとってわかりにくいと思われる所は、最大限にページをさいて詳しい解説を加えてあります。この本を読んでいただければ、今まで何となく良くわからずにウヤムヤになっていた部分がきっと明らかになり、アナタのパソコンライフの未来もバラ色に輝くことでしょう。Z80CPU同様、どうかこの本も、皆さんのパソコンライフの一助としてご活用下さい。

目 次

| | |
|----------------------------|----|
| 第0章 マシン語初体験 / | 1 |
| 0-1 マシン語を見る | 2 |
| 0-2 マシン語モニター | 3 |
| 0-3 マシン語を入力する | 5 |
| 0-4 CPU、ビット、バイト | 7 |
| 0-5 メモリー、番地、16進法 | 9 |
| 0-6 入出力装置 | 12 |
| 0-7 モニターMコマンドのまとめ | 14 |
| 0-8 メモリーの内容を見る | 15 |
| 0-9 マシン語プログラムを実行する | 16 |
| 第1章 プログラム作りに挑戦 / | 19 |
| 1-1 CPUとマシン語 | 20 |
| 1-2 ニーモニックとアセンブラ | 21 |
| 1-3 ALレジスタ登場 / | 24 |
| 1-4 ニーモニック表記のルール | 26 |
| 1-5 ハンドアセンブルの注意点 | 29 |
| 1-6 暴走の恐怖 / | 32 |
| 1-7 プログラムの止め方をマスター / | 37 |
| 1-8 止め方についての注意 | 40 |
| 1-9 BASICインタプリタについて | 41 |
| 第2章 レジスタに挑戦 / | 45 |
| 2-1 全レジスタそろいぶみ / | 46 |
| 2-2 データをレジスタに格納する | 49 |
| 2-3 レジスタの中味を見る | 51 |

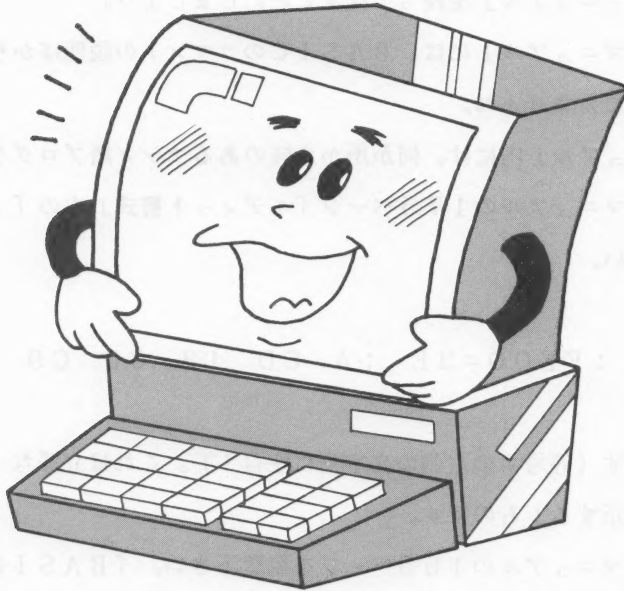
| | | |
|-----|-----------------|----|
| 2-4 | 16ビットレジスタに挑戦 | 55 |
| 2-5 | IXレジスタと間接アドレス指定 | 58 |
| 2-6 | レジスタペア | 61 |
| 2-7 | 再び上下位逆転の注意 | 64 |
| 2-8 | マシン語の構造を見る-1- | 66 |
| 2-9 | マシン語の構造を見る-2- | 68 |

| | | |
|------|-----------------|-----|
| 第3章 | 画面表示に挑戦! | 73 |
| 3-1 | ♥表示にアタック! | 74 |
| 3-2 | VRAMって何? | 75 |
| 3-3 | 表示の要素「何を」 | 77 |
| 3-4 | アトリビュートとは? | 79 |
| 3-5 | OUTコマンドを用いて | 81 |
| 3-6 | I/Oポートの謎 | 83 |
| 3-7 | 入出力命令 | 84 |
| 3-8 | ♥表示をマシン語で! | 86 |
| 3-9 | ♥4個表示をめざして | 88 |
| 3-10 | マシン語での条件判断 | 93 |
| 3-11 | ♥4個表示の完成 | 97 |
| 3-12 | 画面反転プログラム | 101 |
| 3-13 | 問題点の整理 | 107 |
| 3-14 | リロケートブルとは? | 108 |
| 3-15 | 相対ジャンプ命令 | 110 |
| 3-16 | 画面反転をリロケートブルに! | 113 |
| 3-17 | サブルーチンとスタック | 117 |
| 3-18 | スタックポインタの働き | 120 |
| 3-19 | システムの使用するスタック | 122 |
| 3-20 | マシン語フリーエリアの確保 | 125 |
| 3-21 | Gコマンドの解明 | 128 |
| 3-22 | マシン語サブルーチンの実行原理 | 131 |
| 3-23 | 第3章を終えるにあたって | 134 |

| | |
|------------------------------|-----|
| 第4章 マシン語ゲームに挑戦 / | 135 |
| 4-1 ゲームの選定 | 136 |
| 4-2 オールBASIC版作成にあたって | 136 |
| 4-3 ゲーム作成の注意点 | 141 |
| 4-4 マシン語化にあたって | 143 |
| 4-5 BASICとマシン語のリンクの実際 | 144 |
| 4-6 BASICのCALL命令 | 145 |
| 4-7 USR関数の理解をめざして-1- | 146 |
| 4-8 USR関数の理解をめざして-2- | 150 |
| 4-9 マシン語サブルーチンの配置 | 153 |
| 4-10 サブルーチンの仕様を決める / | 155 |
| 4-11 移動方向決定ルーチンの具体化 | 158 |
| 4-12 キャラクター表示ルーチンの具体化 | 163 |
| 4-13 ワークエリアの設定 | 166 |
| 4-14 データ引き渡しの仕様 | 167 |
| 4-15 サブルーチンSARKMVの作成 | 173 |
| 4-16 サブルーチンTRONMVの作成 | 175 |
| 4-17 移動用下位ルーチンの作成 | 176 |
| 4-18 表示ルーチンCHRPRの作成 | 184 |
| 4-19 キャラクター選定サブルーチンCHRDETの作成 | 186 |
| 4-20 プリントルーチンPRINTの作成 | 190 |
| 4-21 画面反転サブルーチン(CREV)の配置 | 191 |
| 4-22 チェックサムの使い方 | 192 |
| 4-23 DATA文に直してアスキーセーブ / | 200 |
| 4-24 トロンゲームマシン語版の完成 / | 201 |
| 4-25 DISKBASICをお使いの方へ | 209 |
| 4-26 最後のコーヒータ임 | 211 |

| | |
|--------------------------------------|-----|
| あとがき | 213 |
| 付録1 Z80命令表 | 219 |
| 付録2 1バイト符号付16進数表 | 251 |
| 付録3 チェックサム プログラムリスト | 252 |
| 付録4 マシン語DATAジェネレーター プログラムリスト | 254 |
| 付録5 トロンゲーム マシン語サブルーチン アセンブラソースリスト | 256 |
| 付録6 XI、XIC、XID マニュアル頁対応表 | 267 |
| 索引 | 268 |

■第0章 マシン語初体験！



■第0章 マシン語初体験！

0-1 / マシン語を見る

さあ、シャープX1で、マシン語の勉強を始めましょう。まず、マシン語とは、どんな姿をしているか見てみましょう。

とはいうものの、どうすればマシン語に出会うことができるのでしょうか？ 私たちの手許には、X1の本体とディスプレイテレビ、それから「マニュアル」しかないと仮定いたします。取りあえず、「マニュアル」を繰って探すことにしましょう。

いかがですか？「マニュアル」には、BASICのコマンドの説明ばかり出ていて、なかなか「マシン語」に出会えませんね。

実は、この「マニュアル」内には、何か所か意味のあるマシン語プログラムが出ています。まず1か所目は？ マニュアルの175ページ「エディット書式」中の「《例》」と書かれている部分を御覧下さい。

: F E O O = 3 E ; A C D 13 00 C9

これがマシン語です（等号=の右側の文字列に注目！）。これは立派な「マシン語プログラム」で「文字Aを表示する」ものです。

もう1か所は？ マニュアルの195ページを御覧下さい。「BASICテープのコピー作成方法」の中に、コピー作成プログラムのリストが出ていますね。「なーんだ、BASICじゃないか？」と思ってはいけませんよ。5行にわたるDATA文がありますね。ここに何やら不可思議な文字列が、50個も書かれています。

《BASICテープ・コピープログラム内の文字列》

21、60、FE、01、20、00、CD、41、00、2A
74、FE、ED、4B、72、FE、CD、44、00、3E
01、CD、1B、00、FE、20、20、F7、21、60
FE、01、20、00、CD、3B、00、2A、74、FE
ED、4B、72、FE、CD、3E、00、C3、13、FE

これは、何をかくそう本格的なマシン語プログラムです！ 普通、BASICのプログラムをテープに録音するには、SAVE でよかったのですね。しかし、BASICのシステムテープは、この方法ではコピーできません。このマシン語プログラムは、BASICではできないこと —— BASICのシステムテープ自体のコピー —— を可能にするものです。マシン語って偉大ですね！

私たちは、マニュアルから2か所、マシン語を探しました。共通しているのは、何やら変な英数字（数字の0～9とアルファベットのA～F）が2つずつ並んでいることですね。また、1か所目の方には、さらに「FE00」という4桁の英数字列もありました。これらの正体は、いったい何でしょうか。

今はまだ、前者のプログラムで「文字Aが表示される」理由、後者のプログラムで「システムテープがコピーできる」理由は、きっとチンプンカンプンなことでしょう。しかし本書を読み終えた時、皆様は、これらの不可思議な文字列 —— マシン語 —— の正体を知ることでしょう。そして、今まで、無味乾燥にしか見えなかった文字列が、急に生き生きと「意味の光」を放ち始めるのを経験することでしょう。では、「素晴らしいマシン語の世界」へ向かってスタート！

〔注〕 X1, X1C, X1Dのマニュアル頁対応表を付録6（267頁）に記載しておりますので、ご参照下さい。

0-2 / マシン語モニター

前節において、私たちは「マニュアル」を探して、マシン語プログラムの姿を観察いたしました。では、私たちの周囲にあるマシン語は、たったこれだけなののでしょうか？

答は、NO です。実は、私たちはウンザリする位、マシン語を見ることができるのです。

BASICを走らせている時、画面に Ok という文字が出て、カーソルが点滅していますね。この状態では、私たちは、BASIC言語しか扱うことができません。では、マシン語を扱う（入力したり、実行したり、リストを出したり）にはどのようにすればよいのでしょうか。

う。

実は、BASICの世界からマシン語の世界に通ずる入口があります。BASICコマンドに MON というのがあるのを御存じですか？ マニュアル107ページを御覧下さい。

制御をBASIC内蔵の機械語モニターに移します。

と書いてあります。マシン語に初めて接する方は、MONコマンドは、使ったことがないかもしれません。しかし、これからマシン語の勉強をしていく上で、このコマンドと無縁に過ごすことはできません。本書を通じて、以後何回も登場してきますから、是非使い方を覚えていただきたいと思います。

ものは試し、とにかく実行してみましょう。 MON とキー・インし、CRキー（リターンキー）を押して下さい。

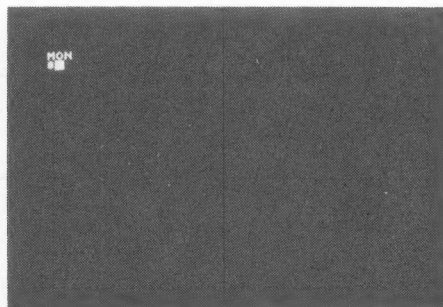


図0-1

普通のBASICコマンドですと、Okが出てからカーソルが点滅するのに、何と星印（*）の後にカーソルが点滅していますね。どうしたことでしょう！？

これは異常ではありません。私達が、BASICの世界をあとにして、マシン語の世界に入った証拠なのです。

マシン語の世界を監視・制御するプログラムをマシン語（機械語）モニターとよびます。モニター（monitor）という言葉はよく聞きますが、辞書で引くと、「監視する、制御する」の意味を持つことがわかります。

普通のBASIC入力状態では、このマシン語モニターは起動されず、「眠って」おります。MONというBASICコマンドは、マシン語モニターを起動するためのものなのです。（以後、マシン語モニターという用語は頻出いたしますので、単にモニターと呼ぶことにします。）

MONを実行し、モニターが起動すると、私たちは、BASICの手を離れて、モニターの管理下に置かれます。これはBASICとは全く違う世界ですから、ユーザーにそのことを知らせるための印が星印*という訳です。この状態のことを、モニターのコマンドレベルともよびます。これに対し、Okが出てカーソルが点滅している状態をBASICのコマンドレベルとよびます。

BASICのコマンドは、普通の英単語に似せて作られていますね。これに対して、モニターのコマンドは、アルファベット1文字で表わされます。X1では、次のコマンドが用いられています。

図0-2 《モニターのコマンド》

| コマンド名 | 機能 |
|-------|-----------|
| M | メモリーセット |
| G | ゴーサブ |
| S | セーブ |
| L | ロード |
| V | ベリファイ |
| D | ダンプ |
| F | ファインド |
| T | トランスファ |
| P | プリンタースイッチ |
| R | リターン |

これら全部をいきなり覚えるのは大変ですから、詳細は、「マニュアル」の175～177ページに任せることにして、基本的なものから使い方をマスターすることにいたします。

0-3 / マシン語を入力する

まず最初にマスターするのは、マシン語の入力方法です。MONによりモニターを起動して下さい。*印が出ましたか？

マシン語の入力には、Mコマンドを用います。次の様にキー・インして下さい。


```

* M D000
: D000=3E
: D001=F3
: D002=01
: D003=F4
: D004=31
: D005=FD
: D006=79
: D007=3E
: D008=03
: D009=01
: D00A=F4
: D00B=21
: D00C=FD
: D00D=79
: D00E=C9
: D00F=00
*

```

1行入力するごとにCRキーを押すと次の行に進みます。最後に、

: D 0 0 F = □

↑カーソル点滅

となりましたら、SHIFT + BREAK を押すと再び*が表示され、モニターのコマンドレベルに戻ります。これがマシン語の入力法です。

まだまだ不明な点が多いと思いますが、もう少しお待ち下さい。

ここで登場したのは、4桁の英数字(D000など)と2桁の英数字(3Eなど)ですね。実は、4桁の方はメモリーの番地(アドレス)を表わし、2桁の方がマシン語を表わしています。

これらについてきちんとした説明を与えるには、まずコンピューターの動作原理を理解しな

ければなりません。ちょうどよい機会ですから、次節から3節にわたり、基礎事項を確認することにいたします。

0-4/CPU、ビット、バイト

コンピュータが1つのまとまった動作をするには、大型機・小型機を問わず次の3つの部分が必要です。

中央処理装置（CPU）

記憶装置（メモリー）

入出力装置（I/O）

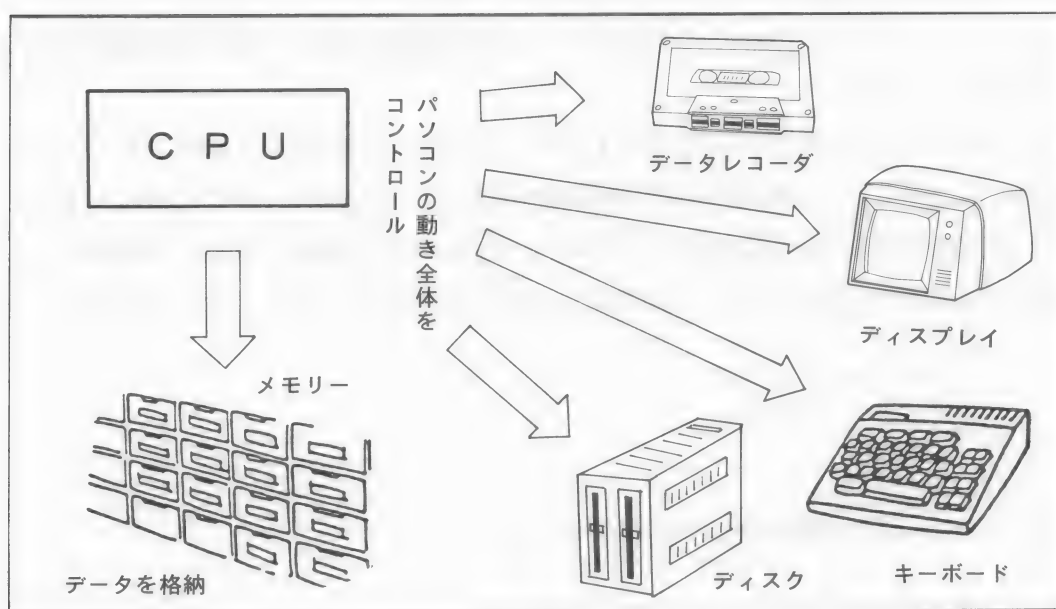


図0-3

まず中央処理装置から見てゆきましょう。英語では、Central Processing Unitといい、頭文字をとって、普通 CPU とよんでいます。

CPUは、コンピュータの中核となる部分で「電子頭脳」とも称せられるように、情報の処理・加工、周辺機器の制御などシステム全般をコントロールします。

SF映画に登場するコンピュータは大型コンピュータで、CPUの部分は、大きな箱状をしています。しかし、最近の飛躍的な半導体技術の発達は、手のひらに載るほどの大規模集

積回路 (LSI=Large Scale Integration)としてCPUを実現することに成功しました。これがマイクロコンピューター (略してマイコン)あるいはマイクロプロセッサとよばれるものです。

パーソナルコンピューターX1の心臓部にもマイコンLSIが搭載されています。

さて、CPUが処理できる情報というのは電気信号の形をとっています。ある基準電圧より、電圧が高いか低いというパターンの組み合わせがCPUのわかる言葉です。これを人間にとって理解しやすくするために、普通、高い状態を1、低い状態を0と表記します。この表記法によると、CPUの処理する情報は、1と0の組み合わせということになります。コンピューターでは、2進法が使われるとよく言いますが、このことを指しているのです。

CPUが処理できる情報の最小単位 (ある信号が1か0かということ) をビット (b i t = binary digitの略)とよびます。1と0をn個組み合わせて得られる情報はnビットであるといいます。

よく、このパソコンは8ビットだとか、16ビットだとかいいますね。これは、パソコンに搭載されているCPUが1度に処理できる情報量を表わしています、X1のCPUは、1度に8ビットの情報処理を行なう能力を持っています。今後、様々な情報は、8ビットを基本にして組み立てられることが多いので、この単位に名前をつけてバイト (b y t e)とよびます。

| |
|-------------|
| 8ビット = 1バイト |
|-------------|

1バイトのデータとは例えば次のような形をしています。

| 第7ビット | | | | 第0ビット | | | |
|-------|---|---|---|-------|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

この各ビットには番号がつけられていて、右から第0ビット、第1ビット、第2ビット、…、第7ビットとよびます。第7ビットを最上位ビット、第0ビットを最下位ビットとよぶこともあります。

1バイトのデータを2進法で表記された数字だと解釈すると、第nビットは 2^n の重みを持つことになります。上の例で言うと

| | | | | | | | | |
|-------|---|-------|---|-------|---|---|-------|---------------------------|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | |
| ↓ | | ↓ | ↓ | | | | ↓ | |
| 2^7 | + | 2^5 | + | 2^4 | + | | 2^0 | = 10進法では |
| | | | | | | | | $128 + 32 + 16 + 1 = 177$ |

となって、この2進数は10進法では177に対応するものになります。

最上位ビットは重みも大きく、ここが1か0かは大きく影響することがあるので、

MSB (Most Significant Bit)

ともよぶことがあります。これに対し、最下位ビットの方は

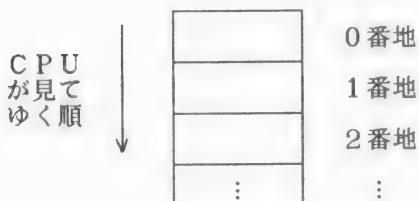
LSB (Least Significant Bit)

ともよべれます。

0ー5／メモリー、番地、16進法

CPUは、メモリーに格納されているプログラムを読み込み、解読することで、所定の動作をする仕組みになっています。

ここでメモリーに眼を転じましょう。メモリーは、CPUが処理する情報（プログラムやデータ）を格納しておく記憶回路で、情報は1バイト単位で区切られて整然と並べられています。各1バイトの区画には番地（アドレス）とよばれる数字が付されています。番地は、0番地、1番地、2番地…という具合につけられていて、CPUは、通常、番地の数字の若い方から順に、メモリーを見てプログラム等を読み込み実行していきます。



さて、X1に搭載されているような8ビット型のCPUでは、番地は16ビットの数値で表わされるように設計されています。（メモリーの番地もCPUにわかる形——2進数——で表わさなくてはなりませんね。）すなわち次のようになります。

| 《2進法》 | | 《10進法》 | |
|------------------|----|--------|----|
| 0000000000000000 | 番地 | 0 | 番地 |
| 0000000000000001 | 番地 | 1 | 番地 |
| 0000000000000010 | 番地 | 2 | 番地 |
| 0000000000000011 | 番地 | 3 | 番地 |
| ⋮ | | ⋮ | |
| 1111111111111110 | 番地 | 65534 | 番地 |
| 1111111111111111 | 番地 | 65535 | 番地 |

従って、メモリーの番地は、0番地から始まり、最大65535番地までつけることができます。

1バイトのデータにしても、また2バイト(=16ビット)で表わされる番地にしても、2進法の0、1の列では私たち人間にはどうもピンと来ませんね。一方、10進法ではコンピュータ側にとって具合が悪い。そこで!と工夫されたのが16進表記法です。すなわち、 $2^4 = 16$ となることを利用し、2進法の数字列を4ビットずつ右から区切って表記していく方法です。各4ビットには次のような記号を対応させます。

| 《2進法》 | | 《10進法》 | | 《16進法》 |
|-------|----|--------|----|--------|
| 0000 | —— | 0 | —— | 0 |
| 0001 | —— | 1 | —— | 1 |
| 0010 | —— | 2 | —— | 2 |
| 0011 | —— | 3 | —— | 3 |
| 0100 | —— | 4 | —— | 4 |
| 0101 | —— | 5 | —— | 5 |
| 0110 | —— | 6 | —— | 6 |
| 0111 | —— | 7 | —— | 7 |
| 1000 | —— | 8 | —— | 8 |
| 1001 | —— | 9 | —— | 9 |
| 1010 | —— | 10 | —— | A |
| 1011 | —— | 11 | —— | B |
| 1100 | —— | 12 | —— | C |
| 1101 | —— | 13 | —— | D |
| 1110 | —— | 14 | —— | E |
| 1111 | —— | 15 | —— | F |

10進法の10～15に相当する部分はまだ数字がありませんから、アルファベットのA～Fをあてることに決められています。

このような16個の（英）数字を用いると、2進数を忠実に反映し、さらに私たち人間にとってもある程度大きさを想像できるような表記法をすることができます。

図0-4 《16進表記法の例》

| 2 進 法 | 16進法 | 10進法 |
|---|---------|-----------|
| $\begin{array}{cc} \boxed{1011} & \boxed{0001} \\ 4\text{ビットずつ区切る} \end{array}$ | B 1 | 1 7 7 |
| $\begin{array}{cccc} \boxed{0101} & \boxed{1111} & \boxed{0011} & \boxed{0111} \\ 4\text{ビットずつ区切る} \end{array}$ | 5 F 3 7 | 2 4 3 7 5 |

16進法と10進法では共通の数字が多くありますから、混同しないように、16進法を銘記するには、数のあとにアルファベットのHをつけることにします。

《例》 5 F 3 7 H, B 1 H

Hは16進法を意味する英語 Hexadecimal の頭文字をとったものです。本書ではこの表記法を採用しますが、他の表記法もあります。

X1の（Hu）BASIC（正式名称はCZ8CB01といいます）では、16進数の頭に&Hをつけています。BASICプログラム中で、16進数を使うにはこの形にしないといけません。

《例》

```
? &H5F37
24375
Ok
```

上の例のようにキー・インすると、5 F 3 7 H に等しい10進数 2 4 3 7 5 が表示されますね。

また、16進数の頭に\$をつける流儀もありますが、この方法は本書では原則として用いません。〔注〕唯一の例外は付録5のリストです。

16進数と、その表記法について、よろしいですか？ 以上まとめておきます。

16進数を表わすのに本書では、
本文中で、数字の後にHをつける
BASICプログラム中で、数字の前に&Hをつける
という両方法を併用し使い分ける。

さて、1バイトの情報は、16進数2桁で表わされますね(00H~FFH)。また、メモリーの番地を表わす2バイトの数は、16進数4桁で表わされます(0000H~FFFFH)。こうして、0-3節に出てきた英数字の正体がわかります。

```
*M D000
: D000=3E
```

たとえば、モニターを起動してからの上のようなメッセージの意味は、メモリーの D000H番地に現在格納されている1バイト情報が 3EH であることを意味しています。他の例についても同様です。

〔注〕モニターによるメッセージでは、16進数しか扱わないので、Hは省略されています。

0-6／入出力装置

さて、コンピューターの3つの大きな部分のうち、CPU、メモリーについて概説しました。最後の入出力装置について見ておきましょう。

先程のモニター起動画面を思い出して下さい。

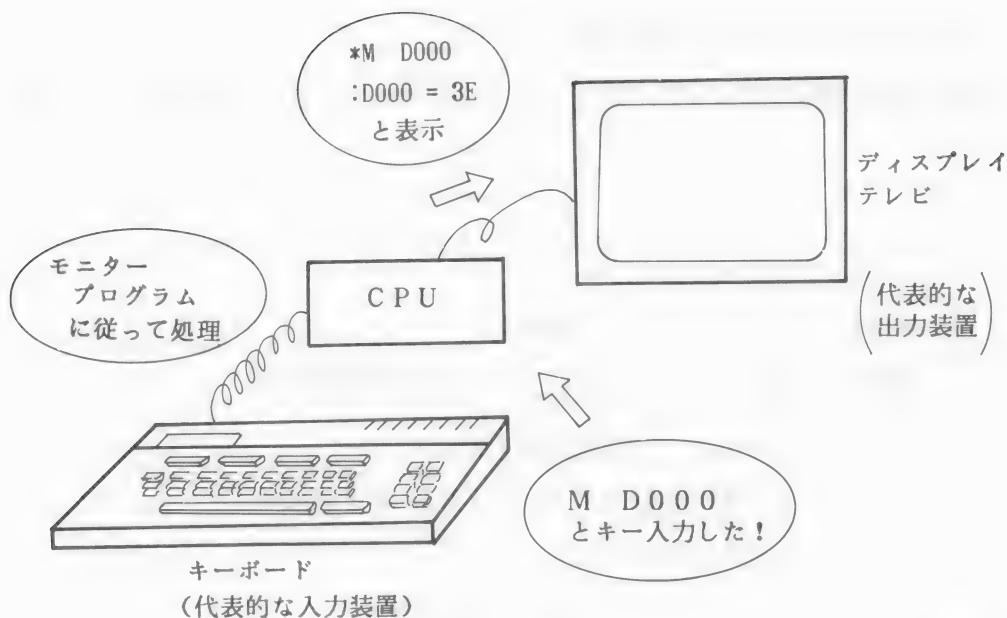


図0-5

パソコンの代表的な入力装置はキーボードです。私たちは、キーボードから M D000 という文字列を入力 (input) しました。この指令はCPUに伝えられ、CPUはメモリーに格納されているモニター・プログラムに従って処理を行ないます。この結果は、代表的な出力装置であるディスプレイテレビの画面に出力 (output) され、D000H番地のメモリーの内容が表示された訳ですね。

このように、入力装置や出力装置は、コンピューターが人間と情報のやりとりをする手・目・耳・口のような働きをいたします。これらを総称して、入出力装置 (Input Output unit) 省略して、I/O などとよびます。



この基本図式をしっかりと頭に入れておいて下さい。

0-7 / モニターMコマンドのまとめ

以上で基礎知識の確認を終りましたので、0-3節に続いて、マシン語入力についてまとめておきます。

| | |
|------|------------------------|
| モニター | Mコマンド |
| 《書式》 | M アドレス (16進数) |
| 《機能》 | 指定したアドレスのメモリー内容を書き換える。 |

[注1] 入力はすべて16進表記で行なう。

16進数を表わす H や &H はこの場合
つけなくてもよい。

[注2] モニターのコマンドレベルに戻るには

SHIFT + BREAK による。

たとえば、モニターを起動して、*M D000 とキー・インしたとすると、「メモリーのD000H番地の内容を書き換える」という指令をCPUに与えたことになります。

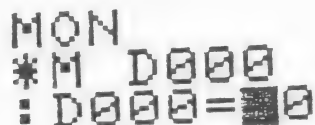


図0-6

すると画面はたとえば上図のようになります。カーソルが点滅している所は、D000H番地に現在格納されている1バイト情報を16進数2桁で表示しています。ですから、3Eとキー・インすることは、D000H番地を3EHという情報に書き換えたことになるのです。0-3節のように次々とメモリー内容を書き換えて、

```

*M D000
:D000=3E
:D001=E3
:D002=01
:D003=F4
:D004=31
:D005=ED
:D006=79
:D007=3E
:D008=03
:D009=01
:D00A=F4
:D00B=21
:D00C=ED
:D00D=79
:D00E=C9
:D00F=00
*

```

図0-7

となるわけですが、これはメモリーの D000H番地～D00EH番地に所定の内容の情報を格納したことになるのです。

では、本当に格納されているかどうかを確認するには、どうすればよいでしょう。それにはモニターのDコマンドを用います。

0-8／メモリーの内容を見る

メモリー内に記憶されている内容を、ディスプレイテレビ等の出力装置へ出力することをダンプ (dump)するといいます。辞書で引くと、「どさりと降ろす」という意味であると出ています。よく、土砂などを運ぶトラックをダンプカーと呼びますね。

さて、dumpの頭文字をとった、モニターのDコマンドが、メモリー内容のダンプをする命令です。

モニター Dコマンド

《書式》 D 開始アドレス 終了アドレス

《機能》 指定範囲のメモリー内容を出力装置にダンプする。

早速、試してみましょう。先程私たちが入力した、D000H～D00EH番地の内容をダ

ンプしてみます。

```
*D D000 D00E
:D000=3E E3 01 F4 31 ED 79 3E />♥.水1y>
:D008=03 01 F4 21 ED 79 C9 00 /...水!y).
*■
```

図0-8

上のように表示されるはずですが、上の一行が、D000H～D007H番地の内容を並べたもの、下一行が D008H～D00FH番地の内容を並べたものです。このように、Dコマンドを実行すると、開始番地から8バイトずつが一行になってダンプされます。

右端の斜線 / の後に不可思議なマークが8個出ていますね。これは今は気にする必要ありませんが、アスキーダンプと言って、メモリー内容の16進数に対応するコード（アスキーコード）を持つ文字・記号を表示しています。詳しくは第3章で学びますが、現段階では「メモリーに格納されている意味ある文字列を探すのに用いる」位に理解しておいて下さい。（たとえば > のコードが 3EH というように読みます）

こうして私たちは、BASICで言えば、プログラムを入力する（Mコマンド）、リストする（Dコマンド）に相当することができるようになったわけですね。最後にプログラムの実行方法について学びましょう。

0-9 / マシン語プログラムを実行する

まだ私は「マシン語」とは何かについて正式に説明しておりません。それなのにプログラムの実行なんて！と思われるかもしれませんが、今はとにかく「初体験コース」の第0章ですから、理屈はわからなくてもやってしましましょう！

先程来、D000H番地から D00EH番地へ格納し、ダンプで確認してみたデータ列は、実は、きちんとした意味ある「マシン語プログラム」です。これらの正体については、次章以降、とくに第3章で詳しく学びますが、本節では、とにかく実行してみることいたします。BASICのプログラムでしたら、RUNなのですが、マシン語の時は？

モニターのGコマンドを用います。

モニター G コマンド

《書式》 G 実行アドレス

《機能》 指定したアドレスから始まるマシン語プログラムを実行する。

準備はよろしいですか？ D000H～D00EH番地には、キチンとデータ（プログラム）が入っていますか？

では *G D000 とキー・インし、CRキーを押して下さい。

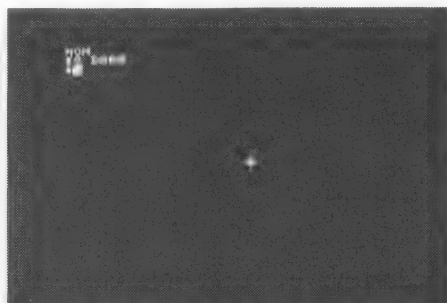


図0-9

正常に動けば（入力ミス等なければ）、画面中央に、赤紫色（マゼンタ）でハートマークが1個表示され、モニターのコマンドレベルに戻るはずです。

そうだったのです。私達が D000H番地から格納したのは、ハートマーク表示プログラムであったのです。

このプログラムの意味を解明するのは第3章のテーマとなります。また、モニターのGコマンドについても、もう少し注意をする必要がありますが、後まわしにして、本章を終える前に、BASICのコマンドレベルに戻る方法について学んでおきましょう。

モニター R コマンド

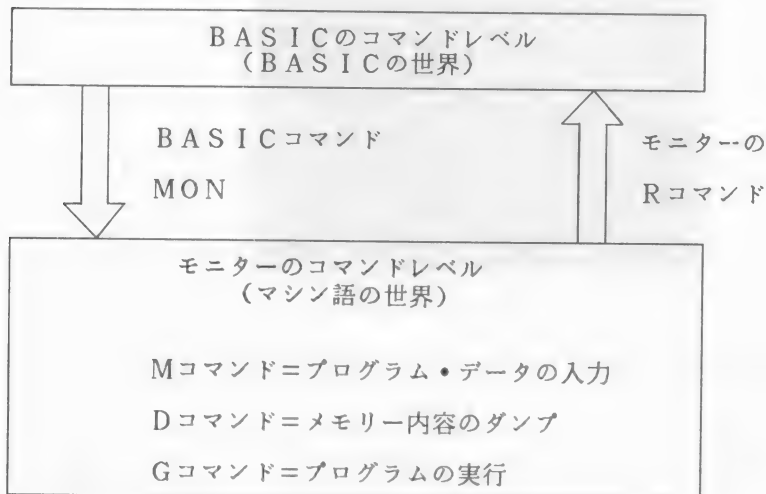
《書式》 R

《機能》 モニターのコマンドレベルからBASICのコマンドレベルへ復帰する。

*R
Ok

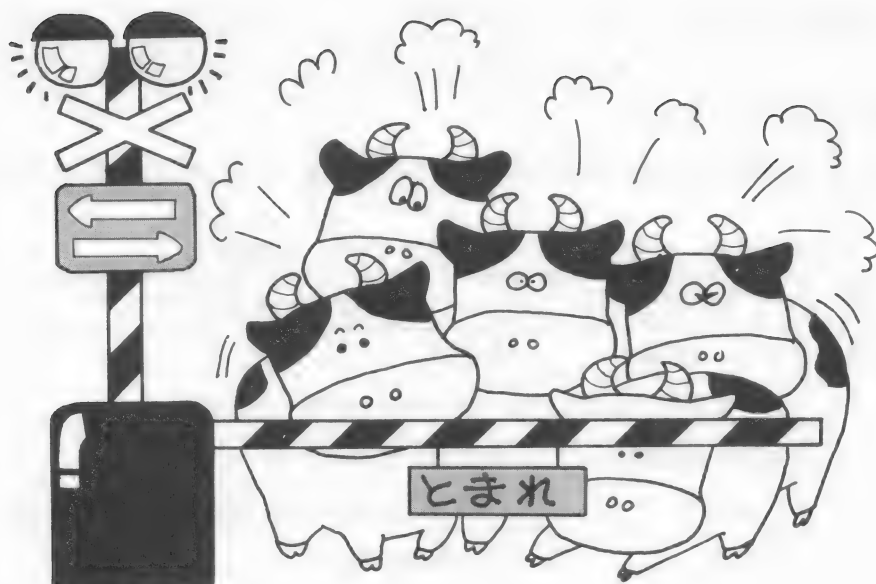
実行すると上のようなになるはずです。Okが表示され、BASICコマンドレベルに戻ったことがわかります。

以上で私たちは次のようなことを行なえるようになりました。



次章以降、私たちは、BASICの世界とマシン語の世界を行きつ戻りつしながら、勉強を進めて行くことになります。これからが、本格的なマシン語の勉強の始まりです。頑張ってください。

■ 第1章 プログラム作りに挑戦！ ■



■第1章 プログラム作りに挑戦！

1-1 / CPUとマシン語

前章において、私たちはCPUの動作原理を確認し、CPUに理解できるのは、メモリーに格納された0と1の組み合わせ（ビットパターンとよぶ）で表わされる情報だけであることを理解しました。

どのようなビットパターンのとき、CPUがどのような動作をするかは、CPUが設計された時点で厳密に決定されています。これを（狭義の、そして本来の意味の）マシン語（機械語）とよびます。ですから、異なった方針の下に設計されたCPU同志では、当然マシン語の体系は異なることになります。

このような立場に立つとき、私たちが対象とする8ビットCPUは3系列に大別されます。

図1-1 《8ビットCPUの分類》

| 系 列 | C P U | 搭載している主なパソコン |
|-------|---|--|
| 80系 | インテル社製 8080、8085 ザイログ社製 Z80 など。 | NEC PC-6000、PC-8000、PC-8800 SHARP MZ-1200、MZ-700、MZ-80B、MZ-2000、MZ-2200 X1(CZ-800C) 東 芝 PASOPIA、PASOPIA7 カシオ FP-1100 |
| 68系 | モトローラ社製 6800、6802、6809 など。 | 富士通 FM-7、FM-8 日 立 ベーシックマスターシリーズ ナショナル JR-100、JR-200 |
| 6502系 | モステクノロジー社製 6502 | アップルII |

これら3系列で、マシン語の体系は異なり、各々を解説するために1冊ずつ本が必要な程です。

さて、私たちのシャープX1は、80系に属するZ80A CPU を搭載しています。「A」なんて余計なものがついている、と不安に思われるかもしれませんが、御安心下さい。Z80 CPU と Z80A CPU とはマシン語は全く同じです。何が違うかというと、CPUを働かせるための水晶発振クロックの周波数が Z80 では2.5MHz（メガヘルツ）、Z80A では4MHzとなっていて、Z80Aの方が1.6倍の速さで動作いたします。（Z80AをZ80のAバージョンとよびます。このような高速版には他に、6MHzのBバージョンがあります。）

本書ではクロック周波数を考慮しなければならない程微妙なプログラムは扱いませんから、Z80AもZ80も区別せず、Z80CPUとして一括して扱うことにいたします。

X1本体の上ぶたをはずして、中をのぞくと、プリント基板上に整然と並べられたLSIやIC（集積回路=Integrated Circuit）たちが見えます。この中に、

| |
|---------------------|
| LH0080A Z80A-CPU |
| SHARP |

と書かれているLSIがありますが、これが心臓部のZ80A CPU です。LH0080A というのは、シャープがZ80A CPU を製造する時の型番で、Z80Aと全く同じものです。

以上の点よろしいですか？　こうして私たちがこれからX1で勉強していくマシン語は、

80系のZ80CPUのマシン語

であることがはっきりいたしました。

1-2/ニーモニックとアセンブラ

Z80CPUは、8ビットCPUですから、そのマシン語も8ビット（=1バイト）を単位に組み立てられています。これらの本来の姿は、8個のビットパターンですが、私たちはこれと等価な2桁16進数として扱うことを学びました。

たとえば前章で、私たちが入力し実行してみたマシン語は次のようなものでした。

| メモリー / アドレス | マシン語 | ビット・パターン |
|-------------|------|-----------|
| D000H | 3EH | 00111110 |
| D001H | E3H | 11100011 |
| D002H | 01H | 00000001 |
| D003H | F4H | 11110100 |
| D004H | 31H | 00110001 |
| D005H | EDH | 11101101 |
| D006H | 79H | 011111001 |
| D007H | 3EH | 00111110 |
| D008H | 03H | 00000011 |
| D009H | 01H | 00000001 |
| D00AH | F4H | 11110100 |
| D00BH | 21H | 00100001 |
| D00CH | EDH | 11101101 |
| D00DH | 79H | 011111001 |
| D00EH | C9H | 11001001 |

図 1 - 2

たしかに、ビットパターンよりは 16 進表示の方が見やすいのですが、しかし、いくら 16 進数を眺めていても、これが CPU にどのような動作を指令するマシン語なのかわかりませんね。そこで、各マシン語には、その内容を連想させる暗記用の名前がつけられています。これを ニーモニック (mnemonic) とよびます。辞書には「記憶を助ける」と出ています。

暗記用の名前だから各人が勝手につけてもよさそうなのですが、これでは混乱をきたしますので、CPU を開発設計したメーカーにより、ニーモニックのつけ方はきちんと決められています。私たちはザイログ社の Z80 CPU を対象としていますので、ザイログ社仕様のニーモニックを用いることになります。

図 1 - 2 で例にあげたマシン語には、次のようなニーモニックが対応します。

| メモリー / アドレス | マシン語 | |
|-------------|------|--------------------------|
| D000H | 3EH |]--- ニーモニック LD A, 0E3H |
| D001H | E3H | |
| D002H | 01H |]--- ニーモニック LD BC, 31F4H |
| D003H | F4H | |
| D004H | 31H |]--- ニーモニック OUT (C), A |
| D005H | EDH | |
| D006H | 79H |]--- ニーモニック LD A, 03H |
| D007H | 3EH | |
| D008H | 03H |]--- ニーモニック LD BC, 21F4H |
| D009H | 01H | |
| D00AH | F4H |]--- ニーモニック OUT (C), A |
| D00BH | 21H | |
| D00CH | EDH |]--- ニーモニック RET |
| D00DH | 79H | |
| D00EH | C9H | |

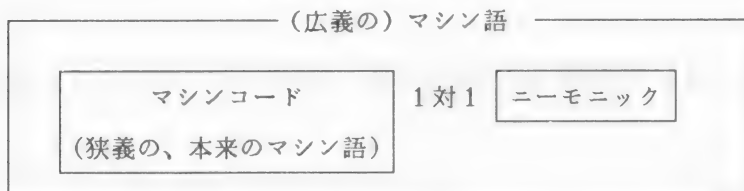
図 1 - 3

マシン語は、1バイトを単位として組み立てられていると述べましたが、1つのまとまった意味をもつマシン語は、1バイト～4バイトで表わされています。その各々に、1つのニーモニックが対応し、上図はその様子を示しています。たとえば、2バイト命令 `3E E3` は、ニーモニック `LD A, 0E3H` で表わされ、3バイト命令 `01 F4 31` は、`LD BC, 31F4H` で表わされ、1バイト命令 `C9` は、ニーモニック `RET` で表わされるというようになっています。

ニーモニックは処理内容を表わす英単語ないしはその省略形からできています。上の例でいうと、`LD` は `load` の略、`RET` は `return` の省略形です。

このように、マシン語の16進コード(1～4バイト)は、ニーモニックと1対1に対応するようになっていますから、しばしばニーモニック表記の方も(広義の)マシン語とよぶことがあります。本書では、マシン語をなるべく広義の意味に用います。そして、16進コードの方をマシンコードとよぶことにします。

《本書での用語の使い方》



人間にとっては、ニーモニック表記でマシン語プログラムを組んでいった方が、「何をしているのか」がわかり、便利なのですが、`LD A, 0E3H` などと入力してもCPUには理解できません。ニーモニック表記されたプログラムを、CPUに理解できるマシンコードに変換する作業をアセンブル(assemble)とよびます。人間の手で、この作業を行なう場合、ハンドアセンブルといいます。

私たちは本書で、ハンドアセンブルの練習を行なうことになりましたが、少し長いプログラムになると大変な作業になります。そこでアセンブル作業をコンピューターに行なわせることを考えます。このための自動変換プログラムをアセンブラ(assembler)とよんでいます。ニーモニック(マシンコードに変換される)と、マシンコードには変換されないけれどもアセンブラに指示を与える命令(疑似命令という)とからなる体系をアセンブリ言語(assembly language)とよびます。本来は、アセンブリ言語をきちんと学んだ上

で、実際にアセンブラを動かして、マシン語プログラムを作成していくのが望ましい姿でしょうが、現時点でX1においてはまだ満足できるアセンブラがないようです（製品として発売されていないという意味で）。そこで、しかたなく本書では、タププリとハンドアセンブルに挑戦していただくことになります。でも頑張ってください。私もそのように勉強してきたのですから。X1に便利なアセンブラができて、ハンドアセンブルから解放される日が来るのを夢みて、せっせとマシン語の実力を蓄えていきましょう！

1-3/Aレジスタ登場！

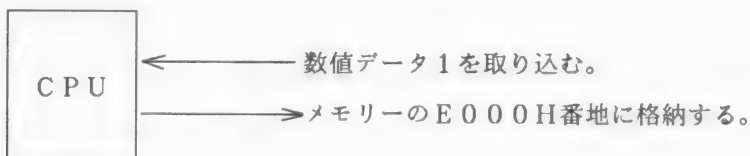
では、いよいよ「正式に」マシン語を勉強していくことにします。用意はよろしいですか。まず、最初に学ぶマシン語は、CPUとメモリーの間データのやりとりに関するものです。

課題1 メモリーのE000H番地に数値データ1を格納すること。

同じことをBASICで実行するのは容易です。POKEというBASICの命令を用いればよいのです。

課題1に対する解答（BASIC版）
POKE &HE000, 1

このような簡単かつ基本的な処理は、マシン語ではどのように行なわれているのでしょうか？ 実は、CPUは、2段階で、この処理を行ないます。



このことを正確に表現するには、レジスタという言葉覚えなくてはなりません。

私は現在、この本の執筆をするため原稿用紙に向かっています。本棚には、コンピューターの本がずらり（？）と並んでいます。本棚はメモリーに相当すると思って下さい。執筆という情報処理を行なう私は、いわばCPUです。さて、私は執筆に必要な資料を参照するために本棚（メモリー）から取り出し、仕事場である机のまわりの手の届く所に一時的に置きます。このような場所にあたるのがレジスタです。自分の手許にデータを置いておけば、本棚（メモリー）に取りに行くより速く処理することができます。一方、そのような場所にあるデータは紛失したり散乱したりするかもしれません。あくまで長期的に保存するには本棚（メモリー）にきちんと整理して格納しておく必要があるのです。

以上、例え話をつかって説明してみました。きちんとすると次のようになります。

レジスタ (register)

メモリーと同様に記憶用の回路であるが、CPU内部であって、データを一時的に記憶しておくために用いられる。

Z80CPU内には、たくさんのレジスタがあります。これらの詳細は次章で学ぶことにし、本章では、Aレジスタについてだけ学ぶことにします。

Aレジスタ

アキュムレータ (accumulator = 累算器)
ともよばれる8ビットレジスタで、CPUが処理するさまざまな算術・論理演算において中心的な役割をはたす。

さて、課題1に戻りましょう。処理手順は正しくは次のようになります。

Aレジスタに数値データ1を取り込む。

Aレジスタの内容（今の場合は1）を、メモリーのE000H番地へ格納する。

もう私たちはマシン語の世界へ片足をつっこみました。上の各処理は、CPUが行なう基本的な処理そのものなのです。ニーモニックでは、次のように表記されます。

```
LD  A, 01H
LD  (0E000H), A
```

この表記と、処理内容を考え合わせると、表記の意味は何となくわかりますね。次節で、この問題をきちんと学びましょう。

1-4 / ニーモニック表記のルール

前節で登場した2つのニーモニックについて考えてみましょう。まず両者共通の LD という部分を取り上げます。

LD は、load の最初と末尾をとった省略形です。BASIC命令にも LOAD というのがあります。「テープからプログラムをロードする」などと使うように、テープやフロッピーディスク等に格納されているプログラム・データをメモリーへ転送することを指しています。load という英単語の原義は「積み込む」という意味ですが、コンピューターではこのように「転送する」という意味あいでも用いられます。(この他に、transfer というものもあります)

従って、LD のつく命令は、データのある場所からある場所へ転送するもので、一般に 転送命令とよばれています。

転送命令では、ちょうど郵便配達を想像するとわかるように、差し出し側と届け先が存在しますね。コンピューター用語では、差し出し側をソース(source=源の意)、届け先をデスティネーション(destination=目的地の意)とよんでいます。

さて、ザイログ社仕様のニーモニックでは次のようなルールがあります。

《ルール1》

| |
|----------------------------|
| 転送命令では、デスティネーション、ソースの順に書く。 |
|----------------------------|

今の例でいうと

| | | | |
|----|-----------------------|---|-------------------|
| LD | <u>デスティネーション</u> A | , | <u>ソース</u> 01H |
| LD | (0E000H) | , | A |

となっています。横書きの時に私たちは左から右へ書き進めますが、データ転送表記の向きは逆であることに、まず注目して下さい。

さて、次のルールです。(0E000H) に関するものです。まず、0E000H の頭の 0 はどんな意味でしょうか？

《ルール2》

16進表記で数を表わすとき、アルファベットのA～Fで始まる数は、その頭に 0 をつける。

これは次章ではっきりとすることですが、レジスタにはAレジスタの他に、B～Fという名前を冠したレジスタもあり、A～Fがレジスタを指すのか、16進数なのか混同しないように区別するためのルールです。

また本書の本文でも採用しているルールですが、

《ルール3》

16進数は末尾に H をつけて表わす。

ニーモニック表記では特にきちんと守る必要があります。

最後に (0E000H) の () について見ておきましょう。

《ルール4》

() を用いると、括弧の中味で指定されたメモリー等の番地を表わす。

これらのルールを知れば、もうニーモニック表記の意味はおわかりですね。

| | |
|----------------|---|
| LD A, 01H | = Aレジスタへ01Hをロードする。 |
| LD (0E000H), A | = メモリーのE000H番地へ Aレジスタにあるデータをロード する。 |

転送命令で1つ確認しておくことがあります。たとえば、LD (0E000H), A
を実行すると、Aレジスタの中味は空になると思いがちですが、そうではないのです。BAS
ICでも、

A = 1

M = A

を実行すると、変数Aの中味1は残り、新しく変数MにAの内容がコピーされて、結局MもA
も中味は1になりますね。これと全く同様です。転送命令では、ソースの中味は不変である
という原則を、よく覚えておいて下さい。

〔注〕 LD A, 01H の 01H の部分に疑問を持たれた読者もおられると思いま
す。01H の頭の 0 は、ルール2で言うところの 0 ではありません。たいていのア
センブラでは、LD A, 1H と書いても、LD A, 1 と 1 を10進表記して
も受けつけてくれるはずです。しかし慣れないうちは、1H とか 1 と書くとは何ビット
データなのか間違えることがあります。Aレジスタは、8ビットのレジスタですから、数の1
はあくまで

ビット・パターン

16進表記

00000001

01H

として格納されるわけです。従って、慣れるまではなるべく、8ビット（後出する16ビット
のときも）のデータで、上位の4ビットが0のときは、0 を省略せずに表記した方がよい
と思います。本書の本文では、この方針で一貫します。

1-5 / ハンドアセンブルの注意点

ニーモニック表記の意味がわかると、次はこれをCPUに理解できるマシンコードに変換する作業をしなくてはなりません。私たちは、ハンドアセンブルにより行ないましょう。

そのためには、マシンコードに翻訳する辞書が必要です。付録の「Z80命令表」がこの役割を果たしてくれます。8ビットロード命令の所を見て下さい。

図1-4 《8ビットロード命令》

| × | | A | B | ----- | n | I | R |
|----|------|----|----|-------|---------|----------|----------|
| LD | A, × | 7F | 78 | ----- | 3E n | ED 57 | ED 5F |
| | ⋮ | ⋮ | ⋮ | ----- | --- | --- | --- |

×印で書かれた最上段の1行は、ソースを表わしています。今の場合、ソースは8ビット数値データですから、 n と記されている所を見ます。

さて、次の2行目は、デスティネーションがAレジスタであるロード命令を集めてあります。この行で、 n の列を見ると、

| |
|---------|
| 3E n |
|---------|

と書かれています。これは、 LD A, n のマシンコードが2バイトであって、 3E n であることを意味しています。私たちの場合、 n は 01H ですから、 LD A, 01H のマシンコードは 3E 01 となるわけです。同様に、Aレジスタに数値 FFH (= 10進法で255) をロードするなら、

| | |
|------------|--------|
| ニーモニック | マシンコード |
| LD A, 0FFH | 3E FF |

となるわけです。

LD A, n の機能は、Aレジスタに8ビット数値をロードするものですが、以後、言葉で説明するかわりに、 $A \leftarrow n$ と矢印を用いて表わすと便利です。このようにして、私たちが初めて学んだマシン語の知識は次のようにまとめられます。

| 解 説 | |
|---------|------------------|
| ニーモニック： | LD A, n |
| | (nは8ビット数値) |
| マシンコード： | 3E n |
| 機 能： | $A \leftarrow n$ |

次は、LD (0E000H), A の番ですね。これをマシンコードに直しましょう。再び「Z80命令表」で8ビットロード命令の項を見ます。LD (nn'), X と書かれた行を注目して下さい。

図1-5 《8ビットロード命令》

| × | A | B | ----- | R |
|-------------|---------------|---|-------|---|
| | | | | |
| LD (nn'), X | 32 n' n | | ----- | |
| | | | | |

今の場合、デスティネーションは指定された番地のメモリーです。 nn' は n を上位8ビット、 n' を下位8ビットとする16ビット数値を表わし、 (nn') は、メモリーの nn' 番地を意味します。この行を横に見ると、1か所を除き、すべて空欄です。すなわち、デスティネーションが (nn') のときは、ソースはAレジスタに限られています。

こうして、 $LD (nn'), A$ というニーモニックに対応するマシンコードは

| |
|------|
| 32 |
| n' |
| n |

すなわち、 $32 n' n$ ということになります。

ちょっと変だと思われた方がおられると思います。 n' と n の書きちがいは？という疑問かと思いますが、これはミスプリではなく、きちんとしたルールなのです。

《上下位逆転の原則》

Z80をはじめ、一般に80系のCPUでは、2バイトの数をアセンブルするとき、その上位1バイト、下位1バイトを逆転させる。

慣れないうちは奇妙な原則に思えるかもしれませんが、80系CPUの設計方針（アーキテクチャ）からするとそれなりの合理性があるようです。

話を戻します。メモリーの番地を指定する nn' は2バイト数値ですから、上下位逆転の原則が適用され、マシンコードには、下位1バイトの n' が先に、上位1バイトの n が後になって変換されるのです。こうして、 $LD (nn'), A$ のマシンコードが、

$32 n' n$ となる訳です。私たちの場合、 $nn'=E000H$ で、 $n=E0$ 、 $n'=00$ ですから、

| ニーモニック | マシンコード |
|------------------|----------|
| $LD (0E000H), A$ | 32 00 E0 |

と変換されます。この上下位逆転則は、いろいろな場面で以後登場してきますから、覚えておいて下さい。

さあ、私たちが2番目に学んだマシン語についてまとめておきましょう。

| 解 説 | |
|---------|---------------|
| ニーモニック： | LD (nn'), A |
| | (nn'は16ビット数値) |
| マシンコード： | 32 n' n |
| 機 能： | (nn') ← A |

1-6 / 暴走の恐怖 /

前節において私たちは初めて、ハンドアSEMBルを経験いたしました。マシンコードにしてみれば、それはCPUに理解できる形ですから、プログラムとしてメモリーに格納し実行してみたいですね。

| ニーモニック | マシンコード |
|----------------|----------|
| LD A, 01H | 3E 01 |
| LD (0E000H), A | 32 00 E0 |

たとえば、メモリーのD000H番地から、このプログラムを格納してみましょう。前章で学んだように、モニターを起動し、次のように入力して下さい。

```

MON
*M D000
: D000=3E
: D001=01
: D002=32
: D003=00
: D004=E0
: D005=00
*

```

図 1 - 6

メモリーの何番地に、どういうデータを入れるか、よろしいですね。しかし、もう少しニーモニックと対応させるとわかりやすくなります。

X1のモニターは優れた画面編集機能（スクリーン・エディット）を持っていて、同様な結果は次のように入力しても得られます。

```

MON
*M D000
: D000=3E01
: D002=3200E0
: D005=00
*

```

図1-7

どうですか？ モニターが自動的に番地を進めてくれますね。このように、ニーモニックと対応するように入力していくと、誤入力を少なくすることができます。

| アドレス | マシンコード | ニーモニック |
|-------|--------|---------------|
| D000H | 3E01 | LD A,01H |
| D002H | 3200E0 | LD (0E000H),A |

これから上のようなリストが続々と出て来ますから、是非慣れていただきたいと思います。

現在作成中のマシン語プログラムに対応することをBASICで書くと次のようになります（Aレジスタを変数Aと見たてています）。

```

10 A=1
20 POKE &HE000,A

```

このBASICプログラムはRUNすれば、もち論正常に動きます。結果を知りたいれば、次のようにすればよいのでしたね。（メモリーの内容を見るには、PEEK関数を用います！）

```

? PEEK(&HE000)
1
OK

```

たしかに、メモリーの E000H番地には、数値1が格納されています。

ですから、私たちのマシン語プログラムも即実行と行きたい所ですが、1つ重大なことを忘れているのです。

モニターのGコマンドで、*G D000 として、私たちのマシン語プログラムを実行すると、CPUは、メモリーの D000H番地に注目し、次のような処理を続けていくことになります。

《CPUの動作》

- ① メモリーの D000H番地にある 3EH をとり込み解読する。
- ② これが LD A, n 型の2バイト命令であることを知り、n を探して次の D001H番地にある 01H をとり込む。
- ③ Aレジスタに 01H をロードする。
- ④ 次の命令を解釈するため、D002H番地にある 32H をとり込み解読する。
- ⑤ これが LD (nn'), A 型の3バイト命令であることを知り、nn' を探して、次の D003H番地と D004H番地の内容を取り込む。
- ⑥ メモリーの E000H番地へAレジスタの内容をロードする。
- ⑦ 次の命令を探しに、D005H番地に注目する。
- ⋮

ここで⑦が重大です。CPUは融通をきかせてくれませんから、プログラムの停止を指示されるまで、このような動作を際限なく繰り返していくことになります。多くの読者はBASICのシステムテープをロードした後に、上の実験をしているでしょうから、メモリー内がクリアされていて異常は起きないと思います。しかし、ゲーム等で遊んでメモリー内に沢山の残渣を残した後に、上のことをしたらどうなるでしょう。一応、D005H番地以降のメモリー内には、私たちにとって未知のデータが格納されていると思わなくてはなりませんね。それでも、CPUはまだプログラムが続いていると認識し、実行していきますから、場合によっては予想もしないことが起こるかもしれません。このような事態を「暴走」とよんでいます。

危険なことです、わざと暴走を起こさせてみます。D005H番地以降の「未知のデータ」として、あらかじめ次のように設定してみてください。

図1-8 《暴走の実験1》

```

MON
*M D005
: D005=C3
: D006=00
: D007=D0
: D008=00
*R
Ok

```

よろしいですか？ 今、入力したものは、「私たちにとって未知のデータ」であるとしてますよ！ さあ、こうして、D000H番地からのプログラムを実行してみてください。いかがですか？ カーソルが消えて、いくらキーを押しても、SHIFT + BREAK さえも効かない状態になってしまいました。これは「未知のデータ」により、プログラムが無限ループに入ってしまったことによる暴走です。

今の場合は、私がわざと設定したいたずらですから、悲劇的な結果にはなりません。マシン語プログラムの暴走は、あくまで「未知な原因」により起こるもので予断はできないのですが、幸いにして軽度な暴走なら直せる場合があります。X1の背面に、リセットボタンがありますね。これを押して下さい。実験1のような軽症の暴走は大抵これで解除され、画面に Ok が表示されて、BASICのコマンドレベルに戻るはずですよ。

しかし、いつもこうだとは思わないで下さい。次に取り返しのつかない暴走例を体験していただきます。今度の「未知のデータ」として、次のように設定してみてください。

図1-9 《暴走の実験2》

```

MON
*M D005
: D005=3E
: D006=1D
: D007=D3
: D008=00
*R
Ok

```

「未知のデータの設定」完了しましたか？ 今度は確実に悲惨な結果になること請け合いますので、プログラム等の壊したくないデータがありましたら、今のうちにテープ等にセーブしておいて下さい。「悲惨な暴走実験」の用意よろしいですか？

では深呼吸をして、モニターを起動し、*G D000 で私たちのプログラムを実行して下さい。いかがですか？ メチャクチャなことが起こったでしょう！ 今回の暴走は、リセットボタンを押しても、IPLが起動するだけです。こうして私たちは、BASICのシステムを破壊し、IPLを呼び出してしまったのです。メモリー内の全プログラム、データはもう取り戻すことはできません。

「暴走」の恐ろしさがおわかりになりましたか？ こうして、私たちは「痛い代償」を払って、大切なことを学んだのです。

《教訓》

- (1) マシン語プログラムの最後には、何らかの実行停止措置を講じなければならない。
- (2) マシン語プログラムを自作する時は、十分なデバッグ（ミスの取り除き）を行なう必要がある。
- (3) 自作のマシン語プログラムを実験的に走らせる前に、なるべくセーブをしておいた方がよい。

この教訓に基づき次節において、私たちはマシン語でのプログラムの停止法を学ばなくてはなりません。

[注] BASICプログラムでは、最後にプログラム終了宣言である END を置かなくても、多くの場合、システムが処理をしてくれて、プログラムは止まるようになっています。しかし、すぐ後に、サブルーチンが続いている時などは、END を置かないと、プログラムは止まらずに、サブルーチンへ飛び込み、RETURN without GO SUB エラーを生じたりします。ですから、BASICプログラムのときも、なるべくEND を書くようにした方がよいのです！

1-7 / プログラムの止め方をマスター！

X1でマシン語を扱う時のプログラムを止める理屈は、実は初心者の方には少し難しい部類に属します。まず、ズバリ答えだけ言ってしまいますと、次のようになります。

Gコマンドにより、マシン語プログラムを走らせる時は、あらかじめプログラムの最後にマシンコード C9 を置いておかねばならない。このようにしておくと、モニターのコマンドレベルに戻り、プログラムは停止する。

まず理屈抜きで実験いたします。次の1バイトプログラムを入力し、実行して下さい。

```
MON
*M D000
: D000=C9
: D001=00
*■
```

図1-10

*G D000 により実行すると、すぐに *印を表示しカーソルが点滅して、モニターのコマンド待ちに戻るはずです。

もう1つ実験をいたします。1-3節以来の懸案であった「E000H番地に1を格納するプログラム」ですが、末尾に C9 を加えて、次の形で入力してみましょう。

```
MON
*M D000
: D000=3E01
: D002=3200E0
: D005=C9
: D006=00
*■
```

図1-11

前節では「暴走」を体験していただいたので、コワゴワでしょうが、今回は安全なことを保証いたしますから、*G D000 で実行して下さい。

いかがですか？ 前の実験と同じく、モニターのコマンド待ちに戻りましたね。つまりプロ

グラムを止めることに成功したのです！

結果を見ましょう。E000H番地に01Hが格納されていることを見るにはどうすればよいのでしょうか？ そう、Dコマンドでダンプすればよいですね。さっそく、*DE000によりE000H番地から、メモリー内容をダンプしてみましょう。

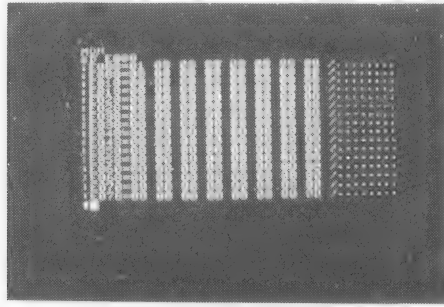


図1-12

いかがですか？ ちゃんと、E000H番地に01Hが格納されていますね。つまり、私たちは、1-3節の課題に対する解答を得たことになるのです。

では、マシンコードC9で表わされるマシン語は、どのような内容のものなのでしょうか？ 本節冒頭で述べたように、実はこの理屈が難しいのです。マシン語サブルーチンの実行の原理とかがわかりますので、詳細は第3章に任せて、本節では簡単に説明しておきます。

マニュアル176ページにモニターのGコマンドについて説明があります。「ゴーサブ」と出ていますね。これがキーポイントです。

BASICでは、ある番地へジャンプさせる命令としてGOTOとGOSUBがあることは御存じだと思います。マニュアルにあるGコマンドの説明は、Gコマンドによるマシン語プログラムの実行方法が、BASICでのGOSUBに相当する形で行なわれるということを言っています。

BASICでは、GOSUB文はサブルーチンの実行を指令する命令でした。そして、サブルーチンの終わりには必ずRETURNが必要でしたね。マシン語においても同様に、Gコマンドでマシン語プログラムを実行すると、そのプログラムはモニターのコマンドレベルから呼び出されるサブルーチンと見なされますので、末尾にはリターン命令が必要なのです。BASICでRETURNを実行すると、サブルーチンを呼び出した次の行に復帰するよう

に、マシン語においてもリターン命令を実行すると、サブルーチンを呼び出したモニターのコマンドレベルに復帰することができます。*印が表示されたのは、この理由によります。

一応、知識としてまとめておきましょう。

解 説

ニーモニック： RET

マシンコード： C9

機 能： マシン語サブルーチンからの復帰

いかがですか？ この説明だけでは、きっとおわかりにならない方もおられると思います。それは当然のことで、真の理解に到達するには、マシン語におけるサブルーチンの実行原理が説明されなくてはなりません。これに関しては、第3章の3-17節以降でタツプリと説明を加えますから、それまでお待ち下さい。わからない方は、現段階では、次のように理解しておいて下さい。

解 説

ニーモニック： RET

マシンコード： C9

機 能： モニターのコマンドレベルに戻ることで、
プログラムを停止させるおまじない。

よろしいですか？ 従いまして、1-3節の課題に対する解答は、次のようになります。

課題1に対する解答（マシン語版）

| アドレス | マシンコード | ニーモニック |
|-------|--------|----------------|
| D000H | 3E01 | LD A, 01H |
| D002H | 3200E0 | LD (0E000H), A |
| D005H | C9 | RET |

1-8 / 止め方についての注意

本節の内容は、少し細かい所に立ち入りますので、初心者の方は読みとばして差しつかえありません。

PC-8001等でマシン語を勉強されたことのある方は、プログラムの止め方として次のものがあることを御存じでしょう。

- ① モニターのホットスタートヘジャンプする方法。
- ② CPUを停止させる方法。

①の場合は、X1の(Hu) BASICでのモニターホットスタート番地は 1000H番地なので、JP 1000H によりここヘジャンプさせれば、モニターのコマンドレベルに戻ってプログラムは停止します。もち論この止め方でも構いません(第3章3-21節参照)。

次に②の場合ですが、マシンコード 76H のHALT命令を実行すれば、CPUは停止(HALT)状態となり、原理的にはプログラムが停止するはずですが、X1の場合は、この方法ではプログラムが止まらなかったり、止まってもリセットボタンでBASICに復帰するとシステムが壊れていたりすることがあります。この原因は複雑でしょうが、おおよそ次のためと思われます。

HALT状態でも、CPUの機能は完全に停止しているのではなく、割り込みを受けつけることができます。ところで、X1においてはそのハードウェアの構造上、キー入力をサブCPUからの割り込みにより処理しています。このために、メインCPUであるZ80には、頻繁に割り込みがかかることになり、HALT状態が解除されてしまうのです。

従って、X1でマシン語プログラムを実行するには、HALTにより停止する方法は適切ではありません。注意して下さい。

本書においては、マシン語初心者の方々を読者として想定しておりますので、当面、モニターの内部構造を利用した止め方 JP 1000H を利用することはせず、マニュアルにあるような RET による止め方を採用いたします。

1-9/BASICインタプリタについて

私たちは、初めて自らの手でマシン語プログラム（短いものですが）を作成し、実行することに成功いたしました。本章を終えるにあたり、今まで曖昧に残してきた「BASICのシステムプログラム」というものについて少し考えておきましょう。

まず、モニターを起動し、*D 0000 9FC4 を実行して下さい。すさまじい速さでダンプリストが画面を下から上に流れていきます（スクロールという）。すべてダンプし終わるのに約3分30秒かかりますが、画面を注目していてわかることは、これらはすべてマシン語であるということです。これが、私たちの日頃親しんでいる「BASIC言語」の真の姿なのです。

私たちは、前章と本章でCPUが理解できるのはマシン語だけであることを知りました。従って、`PRINT "A"` などという文字列は当然CPUには理解不能ですね。ちょうどニーモニック表記をアセンブルしてマシン語に直したように、私たちにとって理解しやすい `PRINT "A"` などの命令も、マシン語に翻訳してやる必要があります。

アセンブラもこのような翻訳プログラムの1つですが、アセンブリ言語の水準はCPUの動作と密着していて、人間がプログラムを作成するには大変な手間がかかります。もっと人間の水準に近づけて、人間が日常使用する言語に近い言葉をマシン語に翻訳できればプログラムはずっと作りやすくなります。このような翻訳プログラムを高級言語とか高水準言語とかよびます。BASIC もその1つですし、他に FORTRAN, COBOL, PASCAL などの言語もそうです。これらの翻訳プログラム自体は、CPUに理解できなくてはなりませんから、先程、BASICの真の姿を見たようにマシン語で書かれています。

さて翻訳の方法には2種類あります。大型コンピューター等で常識的に採用されている方法は、日常言語に近い言語で書かれたプログラムを一挙にマシン語に変換してしまうものです。これをコンパイラとよびます。FORTRAN, COBOL, PASCAL等の言語は普通コンパイラ方式をとっています。

もう1つの方法はプログラムを一挙にマシン語化するのではなく、少しずつ翻訳しては、その内容にあたる処理を行なうマシン語サブルーチン呼び出ししていく方法で、インタプリタとよべれます。インタプリタ方式の代表は BASIC です（最近では LOGO という言語もありますね）。

コンパイラでは、一度マシン語化してしまえば、実行時には完全なマシン語プログラムとな

っていますから高速処理を行なうことができます。一方、インタプリタでは、逐語訳的に翻訳しては実行というパターンを繰り返すために、どうしてもある限度以上のスピードは出せません。よく、BASICでゲームを作るとリアルタイム（プログラムでの時間の進行と現実でのそれが同じであること）処理には不向きだと言われますが、それは、パソコンのBASICがインタプリタであるからなのです。本書で、私たちはマシン語を利用することで、高速ゲーム作りに挑戦いたします。

さて、X1の基本的なシステムプログラムであるBASICインタプリタも巨大なマシン語プログラムであり、実に全メモリーの3分の2近くを占領しています。第0章0-2節で私たちの周囲にウンザリする位マシン語があると述べたのは、こういうわけなのです。

X1では、他の言語も使えるようにメモリーをできるだけ自由な形にしてあります（クリーン設計という）。メモリーには、読み書き自由のRAM（Random Access Memory）と、読みとり専用のROM（Read Only Memory）の2種がありますが、X1は65536バイト分のメインメモリーをすべてRAMにしてあります。よく 64K フルRAM などと言っていますね。コンピューターでは、 $2^{10} = 1024$ のことを K という単位で表わします（キロと区別するためケーと読むそうです）。65536バイト=64Kバイト となるわけですね。

しかし、これでは電源投入時にメモリー内は空で何もできませんから、テープからシステムプログラムを読むための最小限のプログラムはROMに入れて消えないようにしてあります。これをIPL（=Initial Program Loader）とよんでいます。電源投入時に

IPL is looking for a program

とメッセージが出ますが、あれはIPLが動作しているのです。IPLROMはシステムプログラム（BASICインタプリタ等）を読み終わると、役目を終えてメインメモリーから切り離され、メインメモリーはすべてRAMになります。

というわけで、X1においてはBASICインタプリタもRAM上にあり、私たちはこれを書き換えてしまうことが可能です。しかしそのためにはBASICインタプリタの内部を熟知してからでないと、「暴走」の危険がありますね。マニュアルの61ページにある注意はそのためのものです。

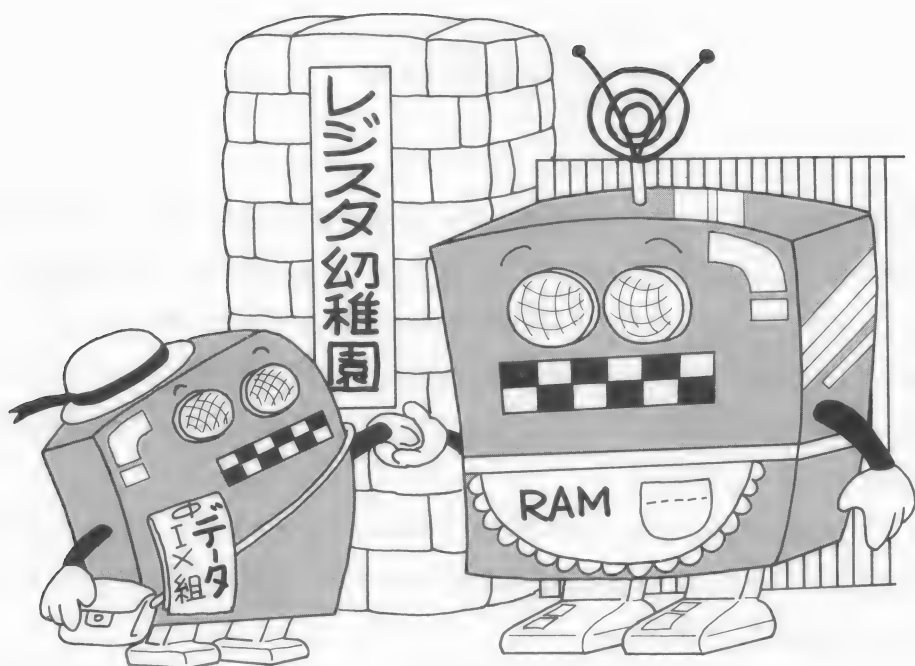
POKEの使用に際しては、BASICプログラムなどメインメモリー内のシステムを破壊してしまう恐れがあるので十分注意して下さい。

BASICインタプリタのシステムプログラムは、メモリー上 0000H～9FC4H番地に置かれています。また、私たちが行番号つきで入力するBASICプログラムは、9FC5H番地以降に格納されます。

前節まで、私たちは、マシン語プログラムをD000H番地から格納しましたが、これはシステムやBASICプログラムを壊さぬようにする配慮のためです。第3章において、私たちは、「マシン語プログラムをどこに置くか」に関して、もっとはっきりしたことを学ぶでしょう。

本章では、Aレジスタとメモリーの情報転送を学びましたが、次章からは、もう少し多彩なプログラムに挑戦いたします。御期待下さい。

■ 第2章 レジスタに挑戦！ ■



■第2章 レジスタに挑戦！

2-1 / 全レジスタそろいぶみ！

前章において、私たちはAレジスタ（アキュムレータ）について学び、初めてマシン語プログラムの作成を体験いたしました。このプログラムはBASICでの次のものに相当していました。

```
10 A=1
20 POKE &HE000,A
30 END
```

このBASICプログラムで、Aは変数ですが、一般にBASICでのプログラム作りでは、変数を上手に使うことは大切でしたね。マシン語プログラムにおけるレジスタの役割もこのようなものです。私たちは、レジスタとその使い方について知っておく必要があります。

Z80CPUの内部には、種々の作業に使うための一時的な記憶回路がたくさんあります。このうち、1ビット単独で働くものはフリップフロップとよばれ、複数ビットを一組にして用いるものをレジスタとよびます。

フリップフロップの例としては、たとえばIFF(Interrupt Flip-Flop＝割り込みフリップフロップ)とよばれるものがありますが、本書の程度を越えますので、Z80CPUの解説書で学んで下さい。

さてレジスタに関してですが、ユーザーに公開されプログラム中で用いることのできるレジスタは全部で22本あります。

次に掲げるレジスタ一覧表で、マス目の数はビット数を表わしています。

16ビットレジスタ…IX, IY, SP, PC

8ビットレジスタ…A, F, B, C, D, E, H, L

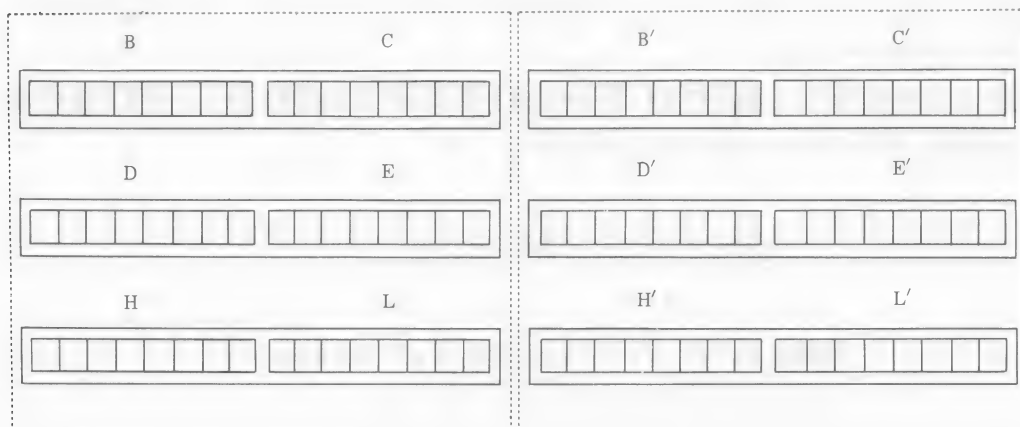
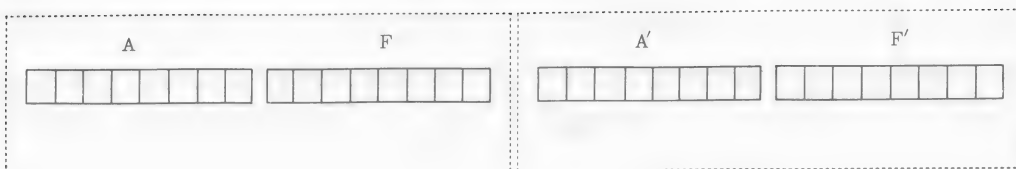
A', F', B', C', D', E', H', L', I

7ビットレジスタ…R

各レジスタには、それぞれ個性があって、よいプログラムを作るには上手に使い分ける必要がありますが、本書を通じてマシン語プログラムを数多く作りながら、だんだんと身につけて

主レジスタ

補助レジスタ



専用レジスタ

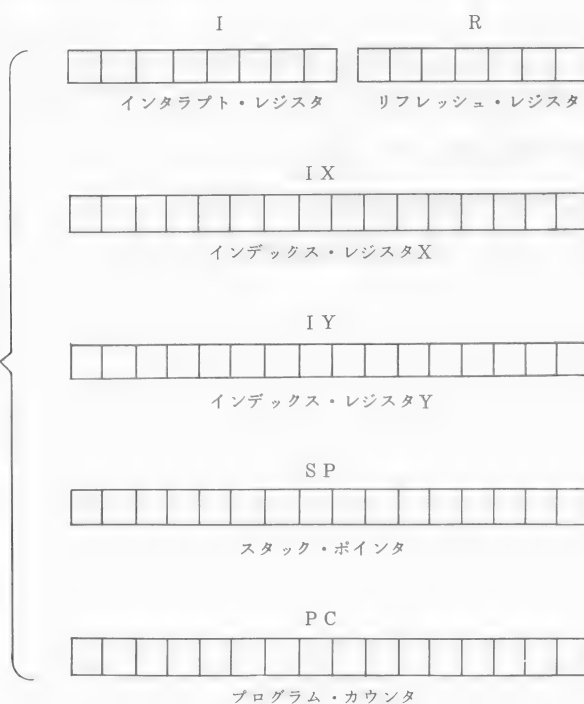


図 2 - 1

いって下さい。

Aレジスタはアキュムレータともよばれることはすでに学びましたね。Fレジスタは、フラグレジスタとよばれ、各ビットに意味をもつレジスタです。B、C、D、E、H、Lの6個のレジスタは汎用レジスタともよばれ、ユーザーが様々な用途に使えるようになっています。以上の8個のレジスタの各々には、同じ機能をもつ補助レジスタが設けられていて、それぞれ'（ダッシュ）をつけて表示します。

専用レジスタとよばれる6本のレジスタは用途が決められていて、ユーザーは用途に従って使わなくてはなりません。

これら22本のレジスタ全部の使い方をいきなり覚えるのは大変ですから、本章では次のレジスタの使い方をマスターすることにいたします。

《本章でマスターするレジスタ》

| | |
|----------------|-------------|
| アキュムレータ..... | Aレジスタ |
| 汎用レジスタ..... | B、C、D、E、H、L |
| インデックス・レジスタ... | IX、IY |

Fレジスタ、スタックポインタ、プログラムカウンタについては、第3章でマスターします。補助レジスタは当面必要ではありません。また、インタラプトレジスタ、リフレッシュレジスタに関しては、本書の程度を越えるので割愛いたします。これらのレジスタの使い方については、読者自らZ80CPUの解説書で勉強して下さい。

以上、よろしいですか？ では、レジスタの勉強スタートです！

2-2 / データをレジスタに格納する

第1章においてと同様に、私たちは次の課題に取り組むことから始めましょう。

課題2 各レジスタに次の8ビット数値を格納するプログラムを作ること。

| | |
|-------|-----|
| Aレジスタ | AAH |
| Bレジスタ | BBH |
| Cレジスタ | CCH |
| Dレジスタ | DDH |
| Eレジスタ | EEH |
| Hレジスタ | 12H |
| Lレジスタ | 34H |

ただし、プログラムはメモリー上 D000H番地から格納するものとする。

まず、ニーモニック表記で考えてみます。前章において、私たちはAレジスタに1バイト（＝8ビット）の数値を格納するロード命令 LD というのを学びました。考え方はそれと全く同様です。次のようになりますね。

ニーモニック

```
LD  A, 0AAH
LD  B, 0BBH
LD  C, 0CCH
LD  D, 0DDH
LD  E, 0EEH
LD  H, 12H
LD  L, 34H
RET
```

よろしいですか？ 私たちはプログラムをモニターのGコマンドで走らせることを前提にしていますから、最後に「プログラムを止めるおまじない」の RET を忘れないで下さいね。

次にこれらをハンドアセンブルで、マシンコードに変換してみましょう。前章でAレジスタに関して行なったのと同様にして、付録の「Z80命令表」を用います。今度は、デスティネーション（データの送り先）がいろいろなレジスタに変わりますから注意して下さい。さて、各ニーモニックに対応するマシンコードは次のようになります。

| 《ニーモニック》 | 《マシンコード》 |
|----------|----------|
| LD A, n | 3E n |
| LD B, n | 06 n |
| LD C, n | 0E n |
| LD D, n | 16 n |
| LD E, n | 1E n |
| LD H, n | 26 n |
| LD L, n | 2E n |

従って、今の場合は、

| マシンコード | ニーモニック |
|--------|------------|
| 3EAA | LD A, 0AAH |
| 06BB | LD B, 0BBH |
| 0ECC | LD C, 0CCH |
| 16DD | LD D, 0DDH |
| 1EEE | LD E, 0EEH |
| 2612 | LD H, 12H |
| 2E34 | LD L, 34H |
| C9 | RET |

図2-2

のようなマシンコードになりますね。最後にこれらをメモリーの D000H番地から配置してできあがりです。

課題2の解答

| アドレス | マシンコード | ニーモニック |
|-------|--------|------------|
| D000H | 3EAA | LD A, 0AAH |
| D002H | 0688 | LD B, 088H |
| D004H | 0ECC | LD C, 0CCH |
| D006H | 16DD | LD D, 0DDH |
| D008H | 1EEE | LD E, 0EEH |
| D00AH | 2612 | LD H, 12H |
| D00CH | 2E34 | LD L, 34H |
| D00EH | C9 | RET |

さっそくモニターを起動し、*M D000 により、プログラムを D000H番地から格納しましょう。

```

MON
*M D000
: D000=3EAA
: D002=0688
: D004=0ECC
: D006=16DD
: D008=1EEE
: D00A=2612
: D00C=2E34
: D00E=C9
: D00F=00
*

```

図2-3

では *G D000 により実行してみましょう。いかがですか？ *印が表示され、カーソルが点滅し、モニターのコマンドレベルに戻ってプログラムは無事停止しましたね。結果を確かめましょう。はて？ そのためにはどのようにすればよいのでしょうか？

2-3/レジスタの中味を見る

私たちは前節で課題2を解決し、レジスタにデータを格納することに成功しました（そのはずですね）。そして、結果を確認するため「レジスタの中味を見る」問題に直面しています。

ところが……本節の標題に反するようですが、レジスタの中味を直接に見ることは実は不可能なのです。付録の「Z80命令表」をいくら探しても、そのようなマシン語はありません。困ったことですね。では結果の確認はできないのでは！？

御安心下さい。レジスタの中味は、直接には見ることはできませんが、間接的に見ることができるからです。第1章での課題1を思い出して下さい。そう！ メモリーに格納すればよいのですよ。

しかし、レジスタの内容をメモリーに格納するための命令は

```
LD (nn'), A
```

の形のものしかありませんでしたね。ソースはAレジスタと決まっていた。では、Bレジスタ以降の内容をメモリーに格納するにはどうしたらよいのでしょうか？

もう察しがついていると思いますが、答は意外に簡単です。そう、Aレジスタ経由でメモリーに転送すればよいのです。

かくして、私たちはレジスタ間のデータ転送命令を学ぶべき段階にまいりました。「Z80命令表」の8ビットロード命令の項で、デスティネーションがAレジスタである行を御覧ください。

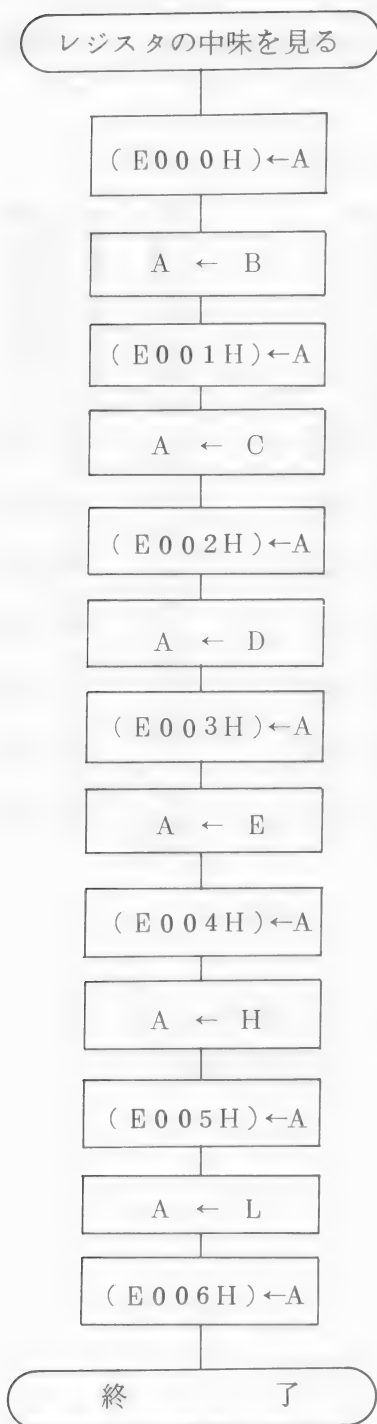
図2-4 《8ビットロード命令》

| | | | | | | | | |
|---------|----|----|----|----|----|----|----|-------|
| × | A | B | C | D | E | H | L | |
| LD A, × | 7F | 78 | 79 | 7A | 7B | 7C | 7D | |
| | | | | | | | | |

となっているはずです。ここに私たちの望む情報のすべてがあります。

データを格納するメモリーの番地は E000H番地からにいたしましょう。処理の手順をフローチャートで示すと次のようになります。

図 2 - 5



同じような処理の繰り返しが見られ、何か工夫する余地がありそうですが、それは後で考えることにして、さっそくプログラムを作ってしまいましょう。課題2のプログラムに続けて配置することになります。

| アドレス | マシンコード | ニーモニック |
|-------|--------|----------------|
| D000H | 3EAA | LD A, 0AAH |
| D002H | 06BB | LD B, 0BBH |
| D004H | 0ECC | LD C, 0CCH |
| D006H | 16DD | LD D, 0DDH |
| D008H | 1EEE | LD E, 0EEH |
| D00AH | 2612 | LD H, 12H |
| D00CH | 2E34 | LD L, 34H |
| D00EH | 3200E0 | LD (0E000H), A |
| D011H | 78 | LD A, B |
| D012H | 3201E0 | LD (0E001H), A |
| D015H | 79 | LD A, C |
| D016H | 3202E0 | LD (0E002H), A |
| D019H | 7A | LD A, D |
| D01AH | 3203E0 | LD (0E003H), A |
| D01DH | 7B | LD A, E |
| D01EH | 3204E0 | LD (0E004H), A |
| D021H | 7C | LD A, H |
| D022H | 3205E0 | LD (0E005H), A |
| D025H | 7D | LD A, L |
| D026H | 3206E0 | LD (0E006H), A |
| D029H | C9 | RET |

図2 - 6

少し長めのプログラムになりましたが、とにかく入力して実行です。できましたか？ 実行が終わったら、 *D E000 により結果を確認します。

図2-7 《レジスタの中味を見る》

```

:E000=AA BB CC DD EE 12 34 00 /エフン. 4.
:E008=00 00 00 00 00 00 00 00 /.....
:E010=00 00 00 00 00 00 00 00 /.....
:E018=00 00 00 00 00 00 00 00 /.....
:E020=00 00 00 00 00 00 00 00 /.....
:E028=00 00 00 00 00 00 00 00 /.....
:E030=00 00 00 00 00 00 00 00 /.....
:E038=00 00 00 00 00 00 00 00 /.....
:E040=00 00 00 00 00 00 00 00 /.....
:E048=00 00 00 00 00 00 00 00 /.....
:E050=00 00 00 00 00 00 00 00 /.....
:E058=00 00 00 00 00 00 00 00 /.....
:E060=00 00 00 00 00 00 00 00 /.....
:E068=00 00 00 00 00 00 00 00 /.....
:E070=00 00 00 00 00 00 00 00 /.....
:E078=00 00 00 00 00 00 00 00 /.....

```

入力ミスがなければ、結果は上図のようになっているはずです。実験成功ですね。こうして、私たちは間接的にはありますが、レジスタの中味を見ることに成功しました。

2-4/16ビットレジスタに挑戦

前節までに私たちは8ビットレジスタの基本的な使い方をマスターしたことになります。

次に、16ビットレジスタであるインデックス・レジスタにアタックしましょう。またまた課題の登場です。

課題3 インデックスレジスタに次の16ビット数値を格納するプログラムを作ること。

IXレジスタ ← 1234H

IYレジスタ ← 5678H

ただし、プログラムはメモリー上 D100H番地から格納するものとする。

付録の「Z80命令表」で、16ビットロード命令の項を御覧下さい。

図2-8 《16ビットロード命令》

| × | AF | | IY | nn' | (nn') |
|----------|----|-------|----|---------------------|---------------------|
| | | | | | |
| LD IX, × | | ----- | | DD 21 n' n | DD 2A n' n |
| LD IY, × | | ----- | | FD 21 n' n | FD 2A n' n |
| | | | | | |

今の場合、デスティネーションはIXまたはIY、ソースは数値nn'ですから次のようになります。

| | |
|------------|------------|
| 《ニーモニック》 | 《マシンコード》 |
| LD IX, nn' | DD 21 n' n |
| LD IY, nn' | FD 21 n' n |

nn' は16ビット (=2バイト) 数値ですから、「上下位逆転の原則」を適用してマシンコードに変換される点を思い起こして下さい。

こうして、課題3の解答は次のようになります。

| 課題3の解答 | | |
|--------|----------|--------------|
| アドレス | マシンコード | ニーモニック |
| D100H | DD213412 | LD IX, 1234H |
| D104H | FD217856 | LD IY, 5678H |
| D108H | C9 | RET |

8ビットレジスタの時と同様に、結果を確認するため、レジスタの内容をメモリーへ転送しましょう。今度は、E100H番地以降に転送することにいたします。

そのための命令を探して、再び16ビットロード命令の項を見ると、次のようなマシン語が見つかります。

| 《ニーモニック》 | 《マシンコード》 |
|--------------|------------|
| LD (nn'), IX | DD 22 n' n |
| LD (nn'), IY | FD 22 n' n |

やはり、アドレスを示す nn' は「上下位逆転」でマシンコードに変換されます。

さあ、プログラムにまとめあげる番ですが、今回注意すべき点は、扱う数値が2バイトですから、IXの内容を格納するのに2バイト分が必要で、E100H番地と E101H番地を要することです。従って、IYの内容は、E102H番地から2バイト分に格納されますね。

これらに注意して、ハンドアセンブルしたものが次のリストです。

| アドレス | マシンコード | ニーモニック |
|-------|----------|-----------------|
| D100H | DD213412 | LD IX, 1234H |
| D104H | FD217856 | LD IY, 5678H |
| D108H | DD2200E1 | LD (0E100H), IX |
| D10CH | FD2202E1 | LD (0E102H), IY |
| D110H | C9 | RET |

よろしいですか？ ではモニターを起動し、*M D100 によりプログラムを格納して下さい。終わりましたら、*G D100 で実行し、*D E100 で結果を確認して下さい。

図2-9 《インデックス レジスタの中味を見る》

```

:E100=34 12 78 56 00 00 00 00 /4.xU...
:E108=00 00 00 00 00 00 00 00 /.....
:E110=00 00 00 00 00 00 00 00 /.....
:E118=00 00 00 00 00 00 00 00 /.....
:E120=00 00 00 00 00 00 00 00 /.....
:E128=00 00 00 00 00 00 00 00 /.....
:E130=00 00 00 00 00 00 00 00 /.....
:E138=00 00 00 00 00 00 00 00 /.....
:E140=00 00 00 00 00 00 00 00 /.....
:E148=00 00 00 00 00 00 00 00 /.....
:E150=00 00 00 00 00 00 00 00 /.....
:E158=00 00 00 00 00 00 00 00 /.....
:E160=00 00 00 00 00 00 00 00 /.....
:E168=00 00 00 00 00 00 00 00 /.....
:E170=00 00 00 00 00 00 00 00 /.....
:E178=00 00 00 00 00 00 00 00 /.....

```

ダンプリストは図2-9のようになるはずですが。何かお気づきになりませんか？ そう！
数字が逆ですね。プログラムミスでしょうか？！

いいえ。正しい結果なのです。常識に反するようですが

| | | |
|--------------|----------|---------|
| E 1 0 0 H 番地 | 3 4 (下位) | } ← I X |
| E 1 0 1 H 番地 | 1 2 (上位) | |
| E 1 0 2 H 番地 | 7 8 (下位) | } ← I Y |
| E 1 0 3 H 番地 | 5 6 (上位) | |

という形で格納されるのです。

これと似たようなことは前にも経験しています。そう、アセンブルにおける「上下位逆転の原則」でしたね。今回は、データ転送における上下位逆転の現象です。すなわち

《データ転送における上下位逆転の原則》

80系CPUにおいては、2バイトデータを扱う命令で、上位1バイトと下位1バイトは逆転される。

とまとめられます。こういった「上下位逆転」は80系CPUのマシン語では常につきまといてきます。慣れないうちは混乱したりしがちですが、図などに書き表わして理解するよう努めて下さい。

2-5 /IXレジスタと間接アドレス指定

インデックスレジスタは、単に16ビットレジスタというだけではなく、独特の機能を持っています。それは、メモリーのアドレスを指定する機能です。

たとえば、次の課題を考えてみて下さい。

課題4 隣接番地のメモリーにデータを格納すること。たとえば

E 2 0 0 H番地 ← データ 0 A H

E 2 0 1 H番地 ← データ 0 B H

E 2 0 2 H番地 ← データ 0 C H

のように。プログラムは、D 2 0 0 H番地から格納するものとする。

この課題の解答は、いろいろ考えられますが、ここではインデックスレジスタ I Xを用いてみましょう（I Yを用いても同様です）。必要とするマシン語は、「Z 8 0 命令表」の8ビットロード命令の項に出ています。

《ニーモニック》

LD (I X + d) , n

《マシンコード》

DD 3 6 d n

ここで n はもちろん数値を表わす1バイトデータですが、 d の方は初登場ですね。d は1バイト数値でディスプレイスメント（displacement = 変位の意）とよばれます。

今、I Xレジスタの内容が E 2 0 0 H であるとします。すると、(I X + d) は、E 2 0 0 H番地からさらに d 番地だけ離れた番地を示すことになります

| 番 地 | | | |
|-----------|--|-------------------|-----------|
| E 2 0 0 H | | … (I X + 0 0 H) | d = 0 0 H |
| E 2 0 1 H | | … (I X + 0 1 H) | d = 0 1 H |
| E 2 0 2 H | | … (I X + 0 2 H) | d = 0 2 H |
| E 2 0 3 H | | … (I X + 0 3 H) | d = 0 3 H |
| E 2 0 4 H | | | |

また、(I X + d) の括弧は I X + d の内容が示すメモリーの番地を表わします。先に出てきた (n n ') の () と同じです。ディスプレイスメントの働き、理解されました

か？ では課題4の解答を与えます。

《課題4のIXレジスタを用いた解答》

| ア ド レ ス | マシンコード | ニーモニック |
|---------|----------------|----------------------|
| D 2 0 0 | DD 2 1 0 0 E 2 | LD IX, 0E 2 0 0 H |
| D 2 0 4 | DD 3 6 0 0 0 A | LD (IX+0 0 H), 0 A H |
| D 2 0 8 | DD 3 6 0 1 0 B | LD (IX+0 1 H), 0 B H |
| D 2 0 C | DD 3 6 0 2 0 C | LD (IX+0 2 H), 0 C H |
| D 2 1 0 | C 9 | RET |

これも *M D 2 0 0 で入力し、*G D 2 0 0 で実行してみてください。結果は、
*D E 2 0 0 E 2 0 2 で確かめます。

《*D E 2 0 0 E 2 0 2 で結果を確認》

:E200=0A 0B 0C 00 00 00 00 00 /.....

このように、インデックスレジスタ等の16ビットレジスタを用いて、アドレスを指定することを間接アドレス指定とよびます。これに対して、(nn')のように16ビット数値でアドレスを指定することを直接アドレス指定とよんでいます。

インデックスレジスタによる間接アドレス指定では、ディスプレイメントを省略することができないこと [(IX) だけではいけません。このときも (IX+0 0 H) とする必要があります。] や、マシンコードがバイト数を多く要することなど欠点がありますが、利点もあります。それは、ディスプレイメントを上手に利用することで、表形式のデータを扱うのに適していることや、また、IXの内容を変えるだけで、データを格納する領域を容易に変更できる点などです。本書では以後、この方法は登場しませんが、有効な利用法を考えてみてください。

2-6/レジスタペア

汎用の8ビットレジスタである B, C, D, E, H, L は次のように組にして、16ビットレジスタとして用いることができます。

BC, DE, HL

これらをレジスタペアとよんでいます。(たとえばBCレジスタペアというようによぶ。ペアを省いてBCレジスタとよぶこともある)

私たちは2-2節の課題2で

Bレジスタ ←———— BBH

Cレジスタ ←———— CCH

というようなことを行ないましたが、B, C を8ビットレジスタとして単独に用いるのではなく、BCレジスタペアとして

BCレジスタペア ←———— BBCCH

という16ビットロード命令と考えることもできます。レジスタペアへの16ビット数値のロード命令には次のものがあります。

《ニーモニック》

LD BC, nn'

LD DE, nn'

LD HL, nn'

《マシンコード》

01 n' n

11 n' n

21 n' n

やはりアセンブル時に上下位逆転が生じていますから注意して下さい。

BCレジスタペアを用いた次の実験を試みてみましょう。

| アドレス | マシンコード | ニーモニック |
|------|--------|----------------|
| D000 | 01CCBB | LD BC, 0BBCCH |
| D003 | 78 | LD A, B |
| D004 | 3200E0 | LD (0E000H), A |
| D007 | 79 | LD A, C |
| D008 | 3201E0 | LD (0E001H), A |
| D00B | C9 | RET |

*M D000 で入力、 *G D000 で実行、 *D E000 E001 で結果の確認をして下さい。

《*D E000 E001 による結果の確認》

:E000=BB CC 00 00 00 00 00 00 /サフ.....

この実験により、LD BC, nn' は、LD B, n と LD C, n' を実行することと等価であることがわかりますね。他のレジスタペアについても全く同様です。ただ、LD BC, nn' のマシンコードは3バイトで済むのに対し、LD B, n と LD C, n' の両方を行なうと4バイト必要ですから、レジスタペアを用いた方が省メモリになります。

レジスタペアはこの他にも重要な機能をいくつか持っています。また BC, DE, HL の各々には個性があります。たとえば、第3章で登場する入出力装置とのデータのやりとりにはBCレジスタペアが活躍するとか、HLレジスタペアは16ビット演算で中心的な役割をはたすことなどがそうです。これらは後で学ぶことにして、本節ではもう1つの重要な機能である間接アドレス指定について考えましょう。

前節で私たちは、インデックスレジスタによる間接アドレス指定について学びましたが、レジスタペアを用いても同様のことができます。すなわち、

(BC) ...BCレジスタペアで示される番地のメモリー

(DE) ...DEレジスタペアで示される番地のメモリー

(HL) ...HLレジスタペアで示される番地のメモリー

という形の間接アドレス指定ができます。

たとえば、HLレジスタペアを E000H、BCレジスタペアを E001H にセットしておくと、E000H番地の内容を E001H番地へ転送するプログラムは以下のよう
に書けます。

《レジスタペアを用いてメモリー間データ転送》

| アドレス | マシンコード | ニーモニック |
|------|--------|---------------|
| D000 | 2100E0 | LD HL, 0E000H |
| D003 | 0101E0 | LD BC, 0E001H |
| D006 | 7E | LD A, (HL) |
| D007 | 02 | LD (BC), A |
| D008 | C9 | RET |

*M D000 でプログラムを入力して下さい。次に、 *M E000 により初期データをセットしておきます（何でも構いません）。

```

*M E000
: E000=BB
: E001=CC
: E002=00
*

```

図2-9

これで実験準備はできました。 *G D000 で実行して下さい。

《*D E000 E001 で結果を確認》

```
:E000=BB BB 00 00 00 00 00 00 /ササ.....
```

結果は上のようになるはずですが、見事に、E000H番地の内容が、E001H番地へ転送されていますね。

メモリー間の転送は直接に行なうことはできません。必ずAレジスタ等のレジスタを用いて（つまりCPUを経由して）行なう必要があることを注意しておきます。

2-7 / 再び上下位逆転の注意

私たちは2-4節において、16ビットレジスタの内容をメモリーに転送する命令

```
LD  (nn'), IX
LD  (nn'), IY
```

を学びました。本節では、16ビットレジスタとして扱った時のレジスタペアについて同様のことを考えておきましょう。

「Z80命令表」の16ビットロード命令を見ると次のマシン語が出ています。

| 《ニーモニック》 | 《マシンコード》 |
|---------------|------------|
| LD (nn'), BC | ED 43 n' n |
| LD (nn'), DE | ED 53 n' n |
| LD (nn'), HL | 22 n' n |

次のプログラムを考えます。

| アドレス | マシンコード | ニーモニック |
|------|----------|-----------------|
| D000 | 01CCBB | LD BC, 0BBCCH |
| D003 | 11EEDD | LD DE, 0DDEEH |
| D006 | 213412 | LD HL, 1234H |
| D009 | ED4300E0 | LD (0E000H), BC |
| D00D | ED5302E0 | LD (0E002H), DE |
| D011 | 2204E0 | LD (0E004H), HL |
| D014 | C9 | RET |

インデックスレジスタの時と同じで、2バイトずつデータを格納しますから、メモリーの番地は、 E000H, E002H, E004H と2バイトおきにする必要がありますね。

さて、これを入力し、実行する前に考えていただきたいのですが、E000H~E005H

番地の6バイト分のメモリー内容はどうなっていますか？ 答えができれば、

*G D000 で実行し、結果を確認して下さい。

《*D E000 E005 で結果を確認》

:E000=CC BB EE DD 34 12 00 00 /フサ⁴...

いかがですか？ 予想通りの結果でしたか？ 答えが合っていた方は、80系CPUにおける「上下位逆転」について基本的な理解ができています。

X1のマシン語においても非常に大切なポイントですから、もう一度説明をしておきます。まず初心者の方が抱くであろう疑問としては（私も初めはそうでした！）、次のことがあろうかと想像いたします。

疑問 LD (nn'), BC は、(nn') ← BC ということだから、
2バイトデータ【BC】を、1バイト分のメモリー【(nn')】に
ロードしていて矛盾である？！

これはニーモニック上の約束ですから、仕方ないのですが、LD (nn'), BC の機能は (nn') ← BC ではありません。正しくは次のようになります。

解 説

ニーモニック：LD (nn'), BC

【nn' は2バイト数値】

マシンコード：ED 43 n' n

機 能：(nn') ← C

(nn' + 1) ← B

たとえば、プログラム中で用いた

LD (0E000H), BC

を実行するとデータの転送状況は次のようになります。

| 《番 地》 | | |
|-------|--|------------|
| E000H | | ← Cレジスタの内容 |
| E001H | | ← Bレジスタの内容 |
| E002H | | |
| | | |

よろしいですか？ レジスタペアのレベルでも、上下位（Bレジスタが上位バイト、Cレジスタが下位バイト）の逆転が起こっていますね。LD (nn'), DE や LD (nn'), HL でも事情は全く同様です。

また、インデックスレジスタでも同じです。IXレジスタの場合をまとめておくと、

解 説

ニーモニック：LD (nn'), IX

マシンコード：DD 22 n' n

機 能：(nn') ← IXの下位バイト
(nn' + 1) ← IXの上位バイト

となっているのです。IYレジスタでも同様です。

以上の注意点、よろしいですか？ 先のプログラムの実行結果の予想が実際と合わなかった方は特に本節を綿密に読み直して理解しておいて下さい。

2-8 / マシン語の構造を見る-1-

以上で、私たちはレジスタやメモリー間のロード命令の基本をマスターしたことになります。小さなものばかりでしたが、いくつかプログラム作りも経験してきました。このあたりで、マシン語の構造についてまとめておくのも有意義なことでしょう。

私たちが会った各マシン語は、1バイト～4バイトのマシンコードで表わされています。これはZ80マシン語のすべてについて言えます。すなわち、

Z80の全命令は、1バイトから4バイトまでのマシンコードで表わされる。

次に各マシン語の内部構造に注目しましょう。マシン語の最初の部分には、それがどのような処理操作をするものかを意味する部分が置かれます。これをオペレーション・コード（略してオペコード、OPコードなどともいう）とよびます。この後には、命令の対象がどこなのか（どの番地のメモリーか、どの数値かなど）を示す部分が続きます。この部分をオペランドとよんでいます。各マシン語は オペコードのみ、あるいは、オペコード+オペランドという構造をとっています。具体的に例を見ていきましょう。

1バイト命令は、オペコードのみ で意味を持ちます。例えば、

《マシンコード》

7 8

《ニーモニック》

LD A, B

は1バイト命令で、7 8 はオペコードです。

2バイト命令は、オペコード+オペランド の形のもと、2バイトで1つのオペコード を意味するものがあります。例えば、

《マシンコード》

3 E n
↑ ↑
↑ オペランド
↑ ↑
↑ オペコード
E D 4 7
↑ ↑
↑ オペコード

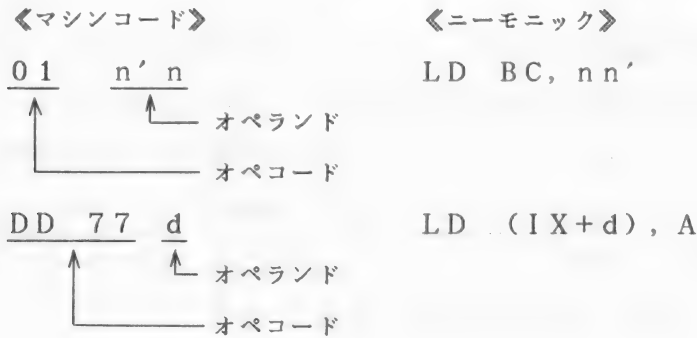
《ニーモニック》

LD A, n

LD I, A (未出)

などがそうです。

3バイト命令は、オペコード+2バイトオペランドの形のものと、2バイトオペコード+オペランドの形のものがあります。例えば、



4バイト命令は、2バイトオペコード+2バイトオペランドの形をとります。例えば、



がその例ですね。

マシン語って多様なようでも、意外に構造が単純であることがわかんと思います。むしろ、このように単純な命令だけで、コンピュータの多様な処理を実行できることを見抜いた天才—— マイクロプロセッサの発明者 —— 的な発想に驚嘆せざるを得ません！

2-9 / マシン語の構造を見る-2-

さらにオペコードの内部構造を見てみましょう。私たちは、いくつかのロード命令に接して、マシンコードの決め方に何か秩序が潜んでいることを感じたはずです。たとえば次のよう

に並べてみると、はっきりしてきます。

| 《マシンコード》 | 《ニーモニック》 |
|----------|----------|
| 7 8 | LD A, B |
| 7 9 | LD A, C |
| 7 A | LD A, D |
| 7 B | LD A, E |
| 7 C | LD A, H |
| 7 D | LD A, L |

気まぐれにマシンコードを決めていたのでは、とてもこんな整然とはしないはずですね。
では、なぜ LD A, B はマシンコード 7 8 を持つのでしょうか。

解答は2進法のビットパターンの中に潜んでいます。16進数 7 8 H を2進法で表示してみましょう。

| 《16進数》 | 《2進数》 |
|--------|-----------------|
| 7 8 | 0 1 1 1 1 0 0 0 |

8ビットですから、0, 1が8個並びますね。

さて、この8ビットを左(MSB)から2ビット、3ビット、3ビットと3つの部分に分けて下さい。

| | | |
|-----|-------|-------|
| 0 1 | 1 1 1 | 0 0 0 |
|-----|-------|-------|

このようになりますね。まず最初の2ビット 0 1 は、LD の部分を意味しています。
すなわち、CPUはマシンコード 7 8 を取り込んだ時、最初の 0 1 により、これが8ビットのレジスタ間転送命令であることを認識するのです。

では、次の3ビットずつは何を意味するのでしょうか。実は、これはレジスタ番号を示しているのです。3ビットでは0～7の8通りの番号を表わすことができますね。8ビットのレジス

タには次のような番号がつけられています。

《レジスタ番号》 (2進法による)

| | |
|---------------|---------------|
| 0 0 0 = Bレジスタ | 1 0 0 = Hレジスタ |
| 0 0 1 = Cレジスタ | 1 0 1 = Lレジスタ |
| 0 1 0 = Dレジスタ | 1 1 0 = (HL) |
| 0 1 1 = Eレジスタ | 1 1 1 = Aレジスタ |

レジスタ番号6 (= 1 1 0) の (HL) というのは異常ですね。これは、レジスタではなく、HLレジスタペアで示される番地のメモリーですから。実は、このことは Z80 CPU の先祖である 8080 CPU の名残りなのです。8080 CPU では (HL) に相当するものを、メモリー上に仮想的に設定された“Mレジスタ”として扱っていたのです。Z80 CPU は互換性を考慮して、8080 CPU の全命令をカバーした上で新機能を追加して設計されたために、このような名残りが見られるのです (2バイトオペコードを持つマシン語は Z80 で追加された新機能です)。

これらのレジスタ番号を、01 の後にデスティネーション、ソースの順に並べるとマシンコードができます。いくつか例を示してみましょう

《ロード命令の構成》

0 1 1 1 1 0 0 0 → 78H は LD A, B
LD A B

0 1 1 1 0 1 1 1 → 77H は LD (HL), A
LD (HL) A

0 1 0 0 1 1 0 0 → 4CH は LD C, H
LD C H

「Z80命令表」で確認すると合っていますね。では、LD (HL), (HL) のような命令は作れるでしょうか？ 同様に考えると、

01 110 110 → 76H
LD (HL) (HL)

マシンコード 76H を持つ命令を探すと、次のように出ています。

解 説

ニーモニック：HALT

マシンコード： 76

機 能：CPU停止命令。

恐ろしいですね！ このことを考えただけでも、メモリーからメモリーへの直接のデータ転送はCPUを一時的にせよ停止させないとできないようになっているのですね。従って、メモリー間のデータ転送は、CPU内のレジスタを経由して行なうようにマシン語が作られているのです。

いかがですか？ 無味乾燥、天の声みたいに思っていたマシンコードが少しでも身近なものになれば、本節の目的は達せられたことになります。

次章からは、いよいよマシン語応用編のスタートです。頑張りましょう！

■ 第3章 画面表示に挑戦！



■第3章 画面表示に挑戦！

3-1 / ♥表示にアタック！

前章までのウォーミング・アップを終えて、私たちはいよいよマシン語による最初の周辺機器制御 —— そうテレビ画面ですよ！ —— に挑戦しましょう！ 当面の課題は、ナン

ト！
テレビ画面への1文字表示
です。

「何を簡単な！」と思われる読者がおられるかもしれません。当然です。BASICでは基本中の基本に属しますからね。例えば、40×25文字表示(WIDTH 40)で、画面中央に♥マークを表示するBASIC文は、

```
LOCATE 20, 12:PRINT "♥"
```

ですね。しかし、これを「テレビ画面という周辺機器の制御」と見る視点こそ、マシン語の視点なのです。この章を読み終えた時、私たちはX1のZ80CPUが、PRINT文にあたる作業をどのように実行するか、その仕組みを知るでしょう。

まず、次のBASIC文を御覧下さい。

```
POKE④ &H31F4, &HE3:POKE④ &H21F4, 7
```

実は、この文は上のPRINT文と全く同じ働きをします。「エッ？」と思われる方がいたら、画面をクリアした後、この文を実行してみてください。おわかりになりましたか？

POKE④ というコマンドに初めて接した読者がおられても、それは当然です。なぜなら、これは「マシン語的」な色彩の強いBASICコマンドだからなのです。

マニュアルの91ページを見ますと、POKE④ の機能は、

VRAM内の指定されたアドレスに1バイトのデータを書き込みます。

(→ OUT)

と書いてあります。「VRAM」という言葉ご存知でしたか？

3-2 / VRAMって何？

VRAMとは、V i d e o R A M の略称で、テレビ画面表示に用いられる特殊な読み書き自由メモリー（RAM）のことです。メモリーである以上、きちんとアドレスがつけられていて、画面の各表示位置と1対1に対応するよう作られています。

シャープX1では、テキスト・モード（PRINT文などで文字を表示する）、グラフィック・モード（CIRCLE文などで図形を表示する）という2つの表示モードがあります。この各々に、VRAMが付随しています。実は、グラフィック用のVRAM（GRAMなどともいう）は、本体には付属しておらず、オプションとして別途購入して、取り付けるようになっています。X1の持つ優れたグラフィック機能を活かすには、不可欠なデバイスなので、まだの方は是非取り付けていただきたいのですが、本章では、お持ちでない方のことも考慮し、テキスト・モードを中心に述べます。（X1C、X1DではGRAMは本体に付属しています。）

テキストVRAMは、 3000H～37FFH の範囲のアドレスを持っています。40×25モードでは、画面は横40個、縦25個のマス目に区切られ、各マス目がVRAMの各アドレスと対応します。図示すると次のようです。X1C、Dをお使いの方は、マニュアルに丁寧な記述があります（X1Cは192ページ～、X1Dは216ページ～）。

| | | | | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|--|--|--|-------|-------|-------|
| 3000H | 3001H | 3002H | 3003H | 3004H | 3005H | 3006H | | | | 3025H | 3026H | 3027H |
| 3028H | 3029H | 302AH | 302BH | 302CH | 302DH | 302EH | | | | 304DH | 304EH | 304FH |
| 3050H | 3051H | 3052H | 3053H | 3054H | 3055H | 3056H | | | | 3075H | 3076H | 3077H |
| 3078H | 3079H | 307AH | 307BH | 307CH | 307DH | 307EH | | | | 309DH | 309EH | 309FH |
| 30A0H | 30A1H | 30A2H | 30A3H | 30A4H | 30A5H | 30A6H | | | | 30C5H | 30C6H | 30C7H |
| 30C8H | 30C9H | 30CAH | 30CBH | 30CCH | 30CDH | 30CEH | | | | 30EDH | 30EEH | 30EFH |
| 30F0H | 30F1H | 30F2H | 30F3H | 30F4H | 30F5H | 30F6H | | | | 3115H | 3116H | 3117H |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| 3370H | 3371H | 3372H | 3373H | 3374H | 3375H | 3376H | | | | 3395H | 3396H | 3397H |
| 3398H | 3399H | 339AH | 339BH | 339CH | 339DH | 339EH | | | | 33BDH | 33BEH | 33BFH |
| 33C0H | 33C1H | 33C2H | 33C3H | 33C4H | 33C5H | 33C6H | | | | 33E5H | 33E6H | 33E7H |

図 3 - 1

[注] マニュアル72ページにもあるように、実は、40×25モードではテキスト画面は2ページ分あります。上図は、1ページ目のもので、全く同様にして、VRAMアドレスの3400H～37E7H が、2ページ目の画面と対応します。

x、y座標から、VRAMアドレスを求めるのは、慣れないうちは大変です。そこで次のようなBASICプログラムを作っておくと便利でしょう。

```
10 REM — X,Y サ`ヒョウ カラ VRAM アドレスヲ モトメル —
20 '
30 INPUT "X,Y = ";X,Y
40 INPUT "ヘ`ージ` ( 1 or 2 ) = ";P
50 IF P<>1 AND P<>2 THEN 40
60 PRINT "VRAM アドレス = ";HEX$( &H3000+(P-1)*&H400+X+Y*40)+"H"
70 PRINT :GOTO 30
```

実行してみましょう。

```
RUN
X,Y = ? 20,12
ヘ`ージ` ( 1 or 2 ) = ? 1
VRAM アドレス = 31F4H

X,Y = ?
```

こうして、x=20、y=12のマス目に対応するVRAMアドレスは、31F4Hであることがわかります。先の POKE④ 文で、

POKE④ &H31F4, &HE3

____部が、このアドレス（16進数）です。

では、すぐ次にある &HE3 とは何でしょう。表示するには、少なくとも「どこに」、「何を」が必要ですが、VRAMアドレスが「どこに」を示しますから、&HE3 は「何を」に対応するものであると察しがつきますね。

3-3 / 表示の要素「何を」

コンピュータでは、文字・記号を規格化されたコード —— ASCIIコード (アスキー・コード) —— により表わします。ASCIIとは、

American Standard Code
for Information Interchange

の頭文字を並べたもので、英数字や記号に対して、1バイト(8ビット=16進2桁)のコードが決められています。日本には、さらにカナ文字がありますから、これもJIS(日本工業規格)により規格化されています。と言いたい所ですが、残念なことに、特にカナ文字に関しては、パソコンの機種により若干の違いがあるようです。他機種からのプログラムの移植に際しては注意が必要です。シャープX1では、マニュアル付属のASCIIコード表のように決められています。

話は戻りますが、第0章0-8節「メモリーの内容を見る」で、アスキーダンプという言葉が出てきたのを覚えていますか。たとえば、モニターを起動した後、* D 0F40 0F D8 とすると次のようなダンプリストが得られます。

```
:0F40=00 00 05 41 55 54 4F 0D /...AUTO.  
:0F48=00 00 00 00 00 00 00 00 /.....  
:0F50=00 00 07 3F 54 49 4D 45 /...?TIME  
:0F58=24 0D 00 00 00 00 00 00 /$......  
:0F60=00 00 03 4B 45 59 00 00 /...KEY..  
:0F68=00 00 00 00 00 00 00 00 /.....  
:0F70=00 00 06 4C 49 53 54 1A /...LIST.  
:0F78=0D 00 00 00 00 00 00 00 /.....  
:0F80=00 00 06 52 55 4E 20 20 /...RUN  
:0F88=0D 00 00 00 00 00 00 00 /.....  
:0F90=00 00 06 4C 4F 41 44 20 /...LOAD  
:0F98=0D 00 00 00 00 00 00 00 /.....  
:0FA0=00 00 06 57 49 44 54 48 /...WIDTH  
:0FA8=20 00 00 00 00 00 00 00 / .....  
:0FB0=00 00 05 43 48 52 24 28 /...CHR$(  
:0FB8=00 00 00 00 00 00 00 00 /.....  
:0FC0=00 00 06 50 41 4C 45 54 /...PALET  
:0FC8=20 00 00 00 00 00 00 00 / .....  
:0FD0=00 00 05 43 4F 4E 54 0D /...CONT.  
:0FD8=00 00 00 00 00 00 00 00 /.....
```

図3-2

ここで右の方にある斜線／から右側がアスキーダンプの部分で、対応するメモリー内容をASCIIコードと見なした文字列を表示しています。これらの文字列はどこかで見た覚えがありますね。そうです、ファンクションキーの内容ですね。

このようにメモリー内容を16進コードで眺めていては見落してしまうような文字列の情報をアスキーダンプの部分が教えてくれるのです。この部分は、BASICシステムプログラムの一部で、ファンクションキーのデータを格納しておく所です。従って、ファンクションキーの内容を変えようと思ったら、この部分に望むデータを入れればよいのです。

アスキーダンプでは、ASCIIコード 00H~1FH の部分はピリオド・を用いて表示しています。これらは、コントロールコードと言って、文字には対応せず、改行したり、画面消去をしたり、ベルを鳴らしたりする動作に対応するコードです。アスキーダンプの説明は以上で終わりです。これでダンプリストの見方は完全にわかりましたね。

再び、私たちが現在直面している問題に眼を転じましょう。ASCIIコード表は次のように使います。

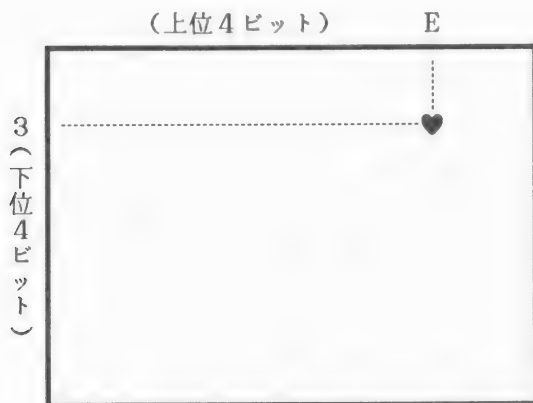


図3-3

ASCIIコード表で ♥マークを探し、図のようにして、♥のASCIIコードが、E3Hであるとわかります。こうして、懸案の

```
POKE④ &H31F4, &HE3
```

が、♥のコード E3H を、VRAMの 31F4H番地に書き込むことだとはっきりしました。

「では！」と、

POKE④ &H31F4, &HE3

とだけ実行してみてください。♥が出ましたか？

キチンと♥が出た方がおられたとしたら、それは運がよいのです。この時にはアドレスを &H3100～&H31FF の範囲でいろいろ変えて実験してみてください。きっと♥以外の不可思議なマークが出るはずですから。

これは、一体どうしたことでしょう？！

3-4／アトリビュートとは？

本章の冒頭部に登場した POKE④ 文をもう一度思い出しましょう。

POKE④ &H31F4, &HE3 : POKE④ &H21F4, 7

マルチステートメントの後半 _____ 部にカギがあるようですね。

結論を申し上げますと、実は画面表示の要素として、「何を」・「どこに」だけでは足りないのです。もう1つ必要です。そう「どのように」なのです！

文字・記号をどのように表示するか（何色か、反転させるか、点滅させるか等）に関わる情報を

アトリビュート (attribute = 属性)

とよびます。X1では、アトリビュートの指定は、1バイト（8ビット）で行ない、8個のビットは各々次の意味を持っています。

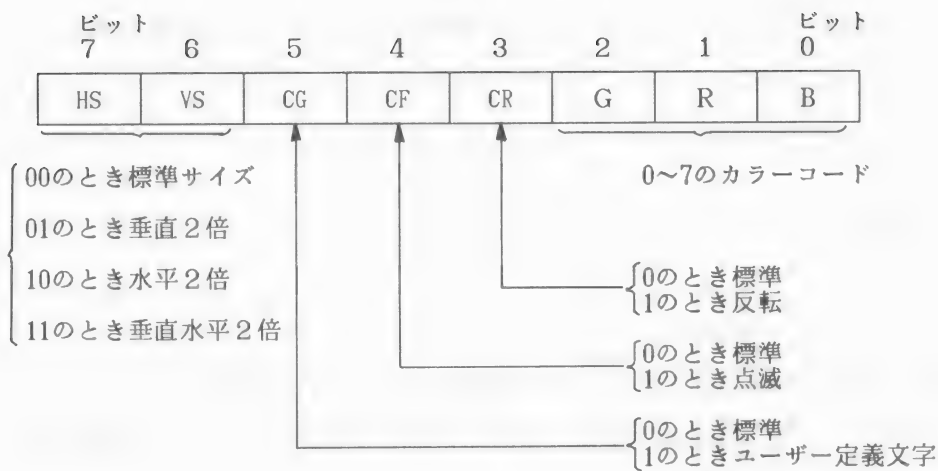


図3-4

VRAMは画面表示用メモリーですから、アトリビュート指定にも当然関与します。VRAMのうち、アトリビュート指定を受け持つ部分を、アトリビュートVRAMとよびます。X1では、アトリビュートVRAMは、2000H~27FFH というアドレスを持っていて、テキストVRAMに格納された文字データ各々にアトリビュート指定できるよう、次のように、テキストVRAMと1対1に対応しています。

| テキストVRAM | アトリビュートVRAM |
|----------|-------------|
| 3000H | 2000H |
| 3001H | 2001H |
| 3002H | 2002H |
| ⋮ | ⋮ |
| 37FEH | 27FEH |
| 37FFH | 27FFH |

対応規則はおわかりですね。テキストVRAMアドレスの先頭の3を2に変えればよいのです。

先程、POKE④ 文の実験で、アドレスをいろいろ変えると♥以外の記号が現われた理由は、そのアドレスのアトリビュートが、垂直水平2倍文字などに指定されていたことによります。ですから、例えば、♥を標準モード、白色（COLOR 7）で表示するのなら、

$$\underbrace{00000}_{\text{標準モード}} \underbrace{111}_{\text{色コード7}} = 10\text{進数で7}$$

とアトリビュート指定しなくてはなりません。これが、

POKE① &H31F4, &HE3: POKE① &H21F4, 7

後半 _____ 部の意味です。

どうですか。ここで、次の公式を確認しておきましょう。

画面表示（テキスト・モード）

= テキストVRAMおよびアトリビュートVRAMの
対応するアドレスに各々のコードを書き込む。

□ コーヒー・ブレイク

実はこの公式、X1に初めて触れた（ニパソコン初体験）私にとって、最初の壁の1つでした。マニュアル168ページ（テキスト画面とその属性ポートへのアクセス方法）は、チンプンカンプンでした。皆様はいかがですか。この公式を理解することは、画面表示のマシン語による制御にとって偉大なる（？）一歩なのです。

3-5/OUTコマンドを用いて

前節で、私たちはテキスト画面の制御要素

どこに = テキストVRAMアドレス
何を = ASCIIコード
どのように = アトリビュートVRAMアドレスと
 アトリビュート指定コード

の理解に到達しました。ここまで来れば、マシン語による画面制御までもう一息です。

頑張りましょう！

私たちは、メモリーとCPUがデータのやりとりをするためのマシン語 —— ロード命令 LD —— を学んできました。VRAMもメモリーの一種ですから、LD命令で！ と思いたい所ですが、ここに最後の関門 —— X1のハードウェアの壁 —— が立ちはだかっています。

再び、POKE④の話に戻りましょう。先にマニュアルの POKE④ の項を見た時、

(→OUT)

と記されていましたね。つまり、同様の結果はBASICの OUTコマンド を用いて次のようにしても得られます。

```
OUT &H31F4, &HE3:OUT &H21F4, 7
```

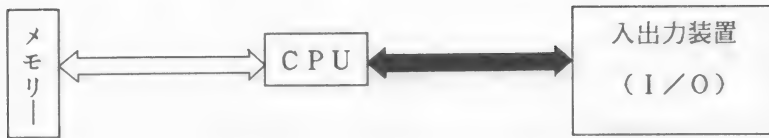
試してみてください。

マニュアル62ページに OUT コマンドについて次のように記されています。

| | |
|----|----------------------|
| 機能 | 出力ポートに1バイトのデータを送ります。 |
|----|----------------------|

ここに何げなく登場している「出力ポート」とは何のことでしょう？

3-6 /I/Oポートの謎



第0章で確認した、コンピュータの基本構成をもう一度、ここで思い起こしましょう。本節のテーマは、図の右の **↔** 部分です。

CPUは、つながれている様々な入出力装置を制御しますが、それは通常、入出力ポート (I/Oポート) というものを間に介して行なわれます。ポート (port) というと、私たちはすぐ「港」を連想しますが、辞書を引くと、「出入口」の意味もあり、コンピュータ用語としての「ポート」は、こちらのようです。



I/Oポートには、メモリーと同様、1バイト単位の区画がアドレスをつけられて整然と並んでいます。I/Oポートのアドレスを I/Oアドレス とよぶことがあります。

8ビットCPUであるZ80では、(メイン)メモリーを最大64Kバイトまで接続できたのでしたね。Z80は、これ以外に256個 ($=2^8$) のI/Oアドレスを持つことができます —— と通常のZ80の本には書いてあります。

シャープのX1では、クリーン設計で、メイン・メモリー64Kバイトを自由に使用するために、画面表示という特殊用途のメモリーであるVRAMをすべてI/Oポートに割りつけてあります。前節までに、しばしば登場した「VRAMのアドレス」は、メイン・メモリー上のアドレスではなく、実はI/Oアドレスであったのです。

ここに重大な疑問があります。テキストVRAMのアドレスは、3000H~37FFHでした。何と2048個ものアドレスがあります。

一方、Z80の普通の本によると、I/Oアドレスは256個だと書いてある。明らかにオ

カシイ！ のです。私も、X1のマシン語を勉強し始めた頃、この矛盾が最大の壁でした。

疑問は、Z80の少し詳しい本を読むことで解決しました。つまり、こういうことなのです。曰ク、

Z80CPUは、メイン・メモリーと同様に

$2^{16} = 65536$ 個のI/Oアドレス指定をする潜在能力を持っている。普通は、下位8ビットのみを使用し、256個のI/Oアドレス指定をする。

と！ X1におけるZ80CPUの使用法は普通ではないのです！ シャープの設計陣が、どのようにして、ハードウェア的にI/Oアドレスの上位8ビットを活かしているのか、私にはわかりませんが、ともかく、このことがX1での入出力制御を理解する最大のキー・ポイントです！

以上をまとめると、X1でのI/Oアドレスは、0000H~FFFFHまでである！ ということになります。このうち、2000H~27FFH番地にアトリビュートVRAM、3000H~37FFH番地にテキストVRAMが割りつけられています。（ついでに述べておくと、4000H~FFFFH番地には、グラフィックVRAMが割りつけられています。）

3-7 / 入出力命令

前節で、私たちは、X1のハードウェア上の特色 —— I/Oアドレスが、メイン・メモリーのアドレスと同じ（64Kバイト）だけある —— を学びました。

Z80CPUが、あるアドレスを指定する時、それが、メイン・メモリーを選択しているのか、I/Oポートを選択しているのか明確にせねばなりません。メモリーとCPUとの情報のやりとり（アクセス）は、ロード命令（LD）というマシン語で行なわれました。Z80では、I/Oポートのアクセスのために、別のマシン語 —— 入出力命令（IN、OUT） —— が用意されています。

本書付録の「Z80命令表」で入出力命令の項を御覧下さい。入力命令12個、出力命令12個の合計24個ありますね。本節では代表として、

入力命令 IN A, (C)
出力命令 OUT (C), A

を取り上げましょう。

まず、OUT (C), A を考えましょう。ロード命令【LD】の時の感覚でいうと、この命令の機能は、

Aレジスタの内容を、
Cレジスタの内容が指定するアドレスの
I/Oポートに出力する。

となりそうですね。[たとえば、LD (HL), A を思い出して下さい!]

しかし、Cレジスタは8ビットのレジスタであり、256個までしかアドレスを区別できません。このことを強調するために、前節でくどい位に説明しておいたのです。

私たちの予想に反して、Z80の少し詳しい本によると、正しくは、次のようになります。

解 説

ニーモニック: OUT (C), A

マシンコード: ED 79

機 能: Aレジスタの内容を、BCレジスタペアの内容が
指定するI/OアドレスのI/Oポートに出力する。

普通は、Bレジスタの指定する上位8ビットのアドレス情報を捨てる使用法をとるため、上記のように、Bレジスタを略したニーモニックが採用されているのでしょう。一般に流通する表記法ではありませんが、X1の特徴を考え、本書では、OUT (C), A の機能を、

I/O (BC) ← A

で表わすことにします。[LD (HL), A の機能を (HL) ← A と表記したの

の連想です。I/Oと付したのはI/Oアドレスを意味します。]

同様に、 IN A, (C) は次のような機能を持ちます。

| 解 説 | |
|---------|--------------|
| ニーモニック： | IN A, (C) |
| マシンコード： | ED 78 |
| 機 能： | A ← I/O (BC) |

これも LD A, (HL) の機能を A ← (HL) と表記したのの連想です。

3-8/♥表示をマシン語で！

大変お待たせいたしました。私たちは、たった♥1個を表示するために、これまで幾多の関門を越えてきました。VRAM, ASCIIコード, アトリビュート, I/Oポート, 入出力命令 —— 理解できましたか。かくして、私たちは、♥表示をマシン語で実行する準備をすべて整えました。

次のマシン語プログラムが、本章冒頭の課題の実現です。(メモリー上、D000H番地からプログラムを格納しました。)

図3-5

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|-------------|-----------------------------------|
| D000 | 3EE3 | LD A,0E3H | POKE@ &H31F4, &HE3 にあたる部分。 |
| D002 | 01F431 | LD BC,31F4H | |
| D005 | ED79 | OUT (C),A | |
| D007 | 3E07 | LD A,07H | POKE@ &H21F4, 7にあたる部分 (アトリビュート出力) |
| D009 | 01F421 | LD BC,21F4H | |
| D00C | ED79 | OUT (C),A | |
| D00E | C9 | RET | …プログラムの停止 |

モニターを起動し、Mコマンドで、D000H番地から、このプログラム(マシンコードの部分)を入力して下さい。終わったら、画面を消去した後、 *G D000 で、実行してみして下さい。

いかがですか？ 画面中央に ♥ が出現し、*で入力待ちとなりましたか？



図3-6

□ コーヒー・ブレイク

お疲れさまです。マシン語では、♥ 1文字の表示のためにも、いろいろなことを理解しておかねばならないことが、わかりましたか？ これをいとも簡単に行なうBASICのPRINT コマンドって偉大だと思いませんか？

実を言うと、私にとっても、ここまでの道は苦難の道だったのです。PC-8001のマシン語の本で、Z80マシン語を勉強していたのですが、PC-8001では、VRAMがメイン・メモリー上にあるので画面表示は、LD命令でよく、X1では、一体どうすればよいのか？ 当時は、X1のマシン語の本もなく、想像を絶する苦闘の末、某ソフト会社のオール・マシン語ゲームの“GAME OVER表示ルーチン”の解析により、理解に到達したのでした。それは、画面中央に GAME OVER と表示するルーチンでした。そこに頻出する ED 79 というマシン・コードは何のためか？ の考察で秘密がわかったのです。私は、感動とともに、私のマシン語勉強ノートにこう書きました。

1983年 △月 ○日
ゲーム「○○○○」の解読により
画面制御への道が開かれた。

さあ、皆さん！ 次のステップへ向かって前進しましょう。

3-9 / ♥4個表示をめざして

余勢をかって、次の課題に取り組みます。

課題 40×25モードで、画面中央から♥を右へ4個表示せよ。
アトリビュートは白色標準モード。

テキストVRAMアドレスは、

座標 (20, 12) (21, 12) (22, 12) (23, 12)

↓ ↓ ↓ ↓

| | | | |
|-------|-------|-------|-------|
| 31F4H | 31F5H | 31F6H | 31F7H |
|-------|-------|-------|-------|

中央 ————— 右

ですから、前節のプログラムを4回繰り返せばよいですね。

図3-7

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|-------------|-----------|
| D000 | 3EE3 | LD A,0E3H | 1つ目の♥を表示 |
| D002 | 01F431 | LD BC,31F4H | |
| D005 | ED79 | OUT (C),A | |
| D007 | 3E07 | LD A,07H | |
| D009 | 01F421 | LD BC,21F4H | |
| D00C | ED79 | OUT (C),A | 2つ目の♥を表示 |
| D00E | 3EE3 | LD A,0E3H | |
| D010 | 01F531 | LD BC,31F5H | |
| D013 | ED79 | OUT (C),A | |
| D015 | 3E07 | LD A,07H | |
| D017 | 01F521 | LD BC,21F5H | 3つ目の♥を表示 |
| D01A | ED79 | OUT (C),A | |
| D01C | 3EE3 | LD A,0E3H | |
| D01E | 01F631 | LD BC,31F6H | |
| D021 | ED79 | OUT (C),A | |
| D023 | 3EE3 | LD A,07H | 4つ目の♥を表示 |
| D025 | 01F621 | LD BC,21F6H | |
| D028 | ED79 | OUT (C),A | |
| D02A | 3E07 | LD A,0E3H | |
| D02C | 01F731 | LD BC,31F7H | |
| D02F | ED79 | OUT (C),A | 4つ目の♥を表示 |
| D031 | 3E07 | LD A,07H | |
| D033 | 01F721 | LD BC,21F7H | |
| D036 | ED79 | OUT (C),A | …プログラムの停止 |
| D038 | C9 | RET | |

上の解答例は、BASICでは、次のようなプログラムに相当します。

```

10 LOCATE 20,12 :PRINT "♥"
20 LOCATE 21,12 :PRINT "♥"
30 LOCATE 22,12 :PRINT "♥"
40 LOCATE 23,12 :PRINT "♥"
50 END

```

これでは、いかにも芸がないですね。また、♥の個数が100個にでもなったらお手上げです。BASICでしたら、FOR ~ NEXT によりループにすることをすぐ思い付きますね。

```
10 FOR X=20 TO 23
20   LOCATE X,12 :PRINT "♥"
30 NEXT X
40 END
```

では、マシン語でループ処理をするには、どうしたらよいでしょう。2つの問題点があります。

- ① VRAMアドレス（BCレジスタ）を1つずつ増やすには？
- ② 4回ループの後、終わりの判定はどうするか？

これらを解決するため、新しいマシン語を学びましょう。

解 説

ニーモニック： INC BC

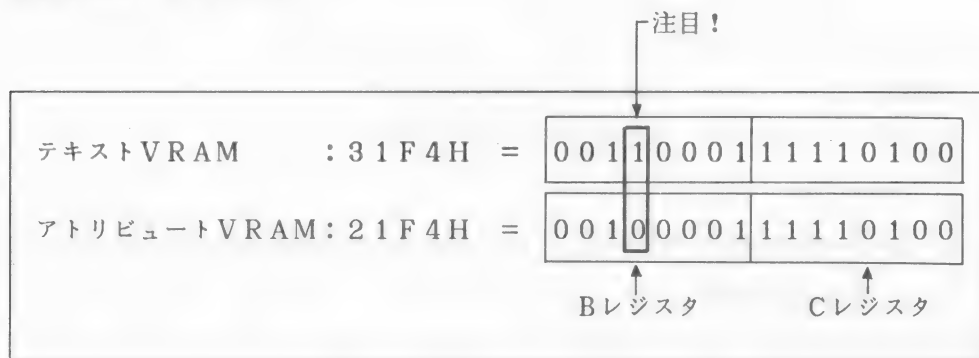
マシンコード： 03

機能： BCレジスタペアの内容を1だけ
増加させる。

[BC ← BC+1]

INCは、increment（増加）からとったものです。INC BC を用いることで、最初に1度だけ、BCレジスタに値 31F4H を入れておけば、あとは1ずつ増やしてゆけばよいのです。これで①は解決！ と思うのは早計です。もう1つ、アトリビュートVRAMも、アドレス指定しなければなりませんでしたね。いろいろな方法があるでし

ようが、次の賢明な方法を紹介します。テキストVRAMと、アトリビュートVRAMのアドレス対応を利用したものです。



例えば、画面中央にあたるVRAMアドレスの対応を御覧下さい。2つの違いは、Bレジスタに格納されるアドレス上位1バイトのうちの第4ビットが 1か0か だけです！

Bレジスタの第4ビット（右から5番目のビット）だけを、1にしたり0にしたりするマシン語はあるか？ と考えたくなります。実はあるのです。本書付録「Z80命令表」の「ビット操作命令」の項を御覧下さい。何と240個もの命令がありますね。しかし、驚くことはありません。これらは、どれか代表を理解すれば、あとは同様ですから。ここでは、次の2つをとり上げます。

ビット操作命令

解 説

ニーモニック: RES 4, B

マシンコード: CB A0

機能: Bレジスタの第4ビットだけを0にする。

(resetする)

解 説

ニーモニック: SET 4, B

マシンコード: CB E0

機能: Bレジスタの第4ビットだけを1にする。

(setする)

これらを用いると、

```
LD BC, 31F4H
```

```
RES 4, B ..... BCは21F4Hになる！
```

```
SET 4, B ..... BCは31F4Hに戻る！
```

となって、テキストVRAMとアトリビュートVRAMを行きつ戻りつできることがわかります。かくして、問題点①は解決しました。

次は②（終わりの判定）です。すぐ思いつくのは次のようなフローチャートです。

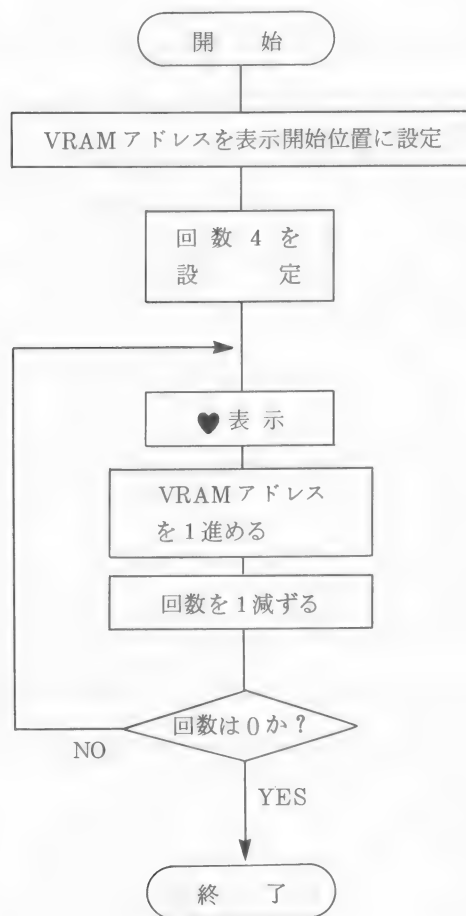


図 3 - 8

条件判断が登場したことに注意して下さい。 BASIC の FOR ～ NEXT ループでも、表には出ませんが、ループ終了の条件判断をしている訳ですね。では、マシン語で条件判断をするには、どうすればよいのでしょうか。

3-10／マシン語での条件判断

話はひとまず、第2章に遡ります。そこでは、8ビット・16ビットの全レジスタが登場していました。このうち、前章で使ってみたのは、A, B, C, D, E, H, L, IX, IYの9個だけです。では、他のレジスタは何のためにあるのでしょうか。これに関しては、おいおい考えていくことにして、本節では、フラグ・レジスタ F を取り上げます。

Fレジスタは、8ビットレジスタですが、他のレジスタと違って、LD命令で値を書き込むことはできません。では、何のためにあるかというと —— そう、本節のテーマである条件判断のために設けられているのです。

フラグ (f l a g) というのは、「旗」のことですね。8本の旗が横一列に並んでいるのを想像して下さい。

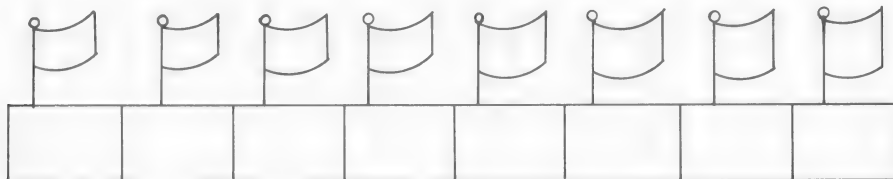


図3-9

これらの旗は、Fレジスタの8個のビットに対応します。ビットが1というのは、旗が立っている状態、ビットが0というのは、旗が降りている状態です。

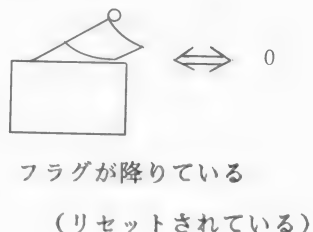
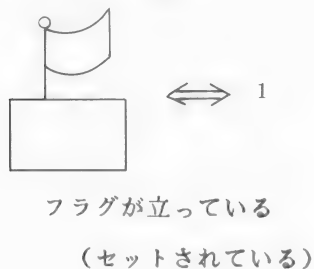


図3-10

これら8個のフラグは、CPUの動作に従って、パタパタとセット、リセットを繰り返すのです。コンピューター用語としてのフラグ、理解できましたか？

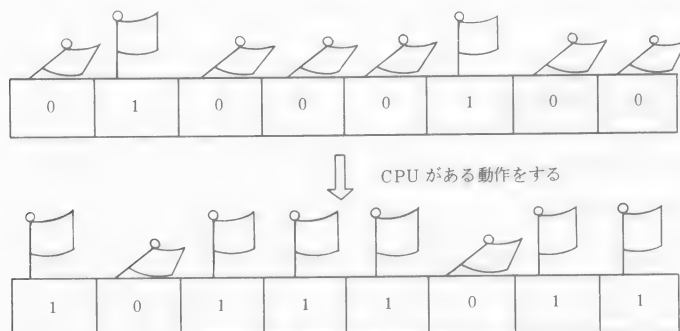
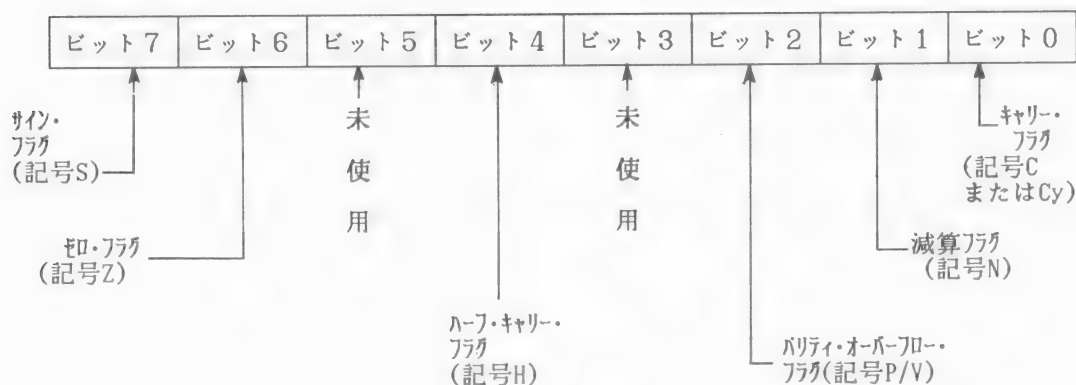


図3-11

さて、8個のフラグ（ビット）には、各々特別な意味があり、その働きに応じて名前がつけられています。



使用される6個のフラグ全部を理解するのは、最初は大変です。フラグ理解の完成は、Z80 CPUの本での読者の勉強に任せることとして、本書では、

ゼロ・フラグ (Z)

キャリー・フラグ (Cy)

の2つだけをマスターしましょう。

先程、Fレジスタは、条件判断のためにあると述べました。条件判断の中で、基本の1つは、2つの数の比較ですね（一方が他方より、大きいか、等しいか、小さいか）。マシン語で、2数の比較を行なうには、どうすればよいのでしょうか。ここでは、2数は8ビットの数といたします（0～255）。

AとBという2数を比較するのに、すぐ思いつくのは、 $A - B$ を計算する方法です。CPUが、この符号（正か負か0か）を判断できれば、大小の比較ができる訳ですね。ここで、注意したいのは、「比較」という目的のためには、 $A - B$ の答の数値自体は必要なく、その符号さえわかればよいということです。

実際、Z80CPU は、「比較命令」を持っていますが、概ね以上のような考え方で作られています。比較に必要なのは、 $A - B$ の符号だと言いました。CPUは、フラグを変化させることにより、その情報を覚えておくのです。

A、Bという8ビットの2数の比較でした。これを、「Aレジスタの内容と、Bレジスタの内容を比較する」と読みかえると、比較命令の1つになります。

解 説

ニーモニック： CP B

マシンコード： B8

機能： $A - B$ により、Aレジスタの内容と、Bレジスタの内容の比較を行なう。ただし、 $A - B$ の答は出力せずフラグのみを変化させる。

● $A - B = 0 \Rightarrow Z$ フラグ = 1

● $A - B \neq 0 \Rightarrow Z$ フラグ = 0

● $A - B < 0 \Rightarrow Cy$ フラグ = 1

● $A - B \geq 0 \Rightarrow Cy$ フラグ = 0

ニーモニックの CP は、compare（比較する）からとったものです。Aレジスタの A が省かれていますね。Z80では、比較される側は、必ずAレジスタでなければならないと、決めてあるからです。これは、アキュムレータとしてのAレジスタが持つ特権の1つです。

次に、フラグ変化ですが、ここでは、ゼロフラグとキャリーフラグに絞って見ることにします（他のフラグも変化しますが、ここでは関知しない立場をとります）。

たとえば、あらかじめ、Aレジスタに 12H（比較されるもの）、Bレジスタに 34H（比較するもの）が、セットされていたとします。ここで、CP B という命令を実行すると、

| ゼロフラグ | | | | | | キャリーフラグ | |
|-------|---|---|---|---|---|---------|---|
| ? | 0 | ? | ? | ? | ? | ? | 1 |

↑
12H ≠ 34H を
意味する。

↑
12H < 34H を
意味する。

のようにフラグが変化します。では逆に、Aレジスタ = 34H、Bレジスタ = 12H で、CP B を実行するとどうでしょう。答は、

| ゼロフラグ | | | | | | キャリーフラグ | |
|-------|---|---|---|---|---|---------|---|
| ? | 0 | ? | ? | ? | ? | ? | 0 |

↑
34H ≠ 12H を
意味する。

↑
34H ≥ 12H を
意味する。

となります。最後に、AもBも 12H であったらどうでしょう。フラグ変化は、

| ゼロフラグ | | | | | | キャリーフラグ | |
|-------|---|---|---|---|---|---------|---|
| ? | 1 | ? | ? | ? | ? | ? | 0 |

↑
12H = 12H を
意味する。

↑
12H ≥ 12H を
意味する。

となります。

比較という操作も、暗黙のうちに減算という演算を行っていますが、このようにCPUが演算を実行したとき、2つのフラグは、

ゼロフラグ : 結果が0か否か、
キャリーフラグ: 加算での桁上げ（キャリー）、減算での借り（BORO）
が、生じたか否か、

という判断のために、主として用いられます。（ $A < B$ のとき $A - B$ を行なうと、8ビット目のもう1つ上の桁から借りが生じますね。ですから、Cyフラグがセットされたのです。）
以上が、比較命令とフラグ変化の実際です。おわかりになりましたか？
行きがけの駄賃に、もう1つ比較命令を覚えておきましょう。CP B と同様です。

解 説

ニーモニック: CP n

マシンコード: FE n (nは8ビット数値)

機能: Aレジスタの内容と、数値nとを比較する。
Aレジスタの内容は変化せず、フラグのみを変化させる。

- $A = n \Rightarrow Z \text{ フラグ} = 1$
- $A \neq n \Rightarrow Z \text{ フラグ} = 0$
- $A < n \Rightarrow Cy \text{ フラグ} = 1$
- $A \geq n \Rightarrow Cy \text{ フラグ} = 0$

3-11/♥4個表示の完成

いよいよ、前々節(3-9)で掲げたフローチャートをマシン語で実現して、♥4個表示プログラムを完成させましょう。

まず、レジスタの役割を決めておきます。

| | |
|----------|--|
| Aレジスタ | : ASCIIコード、アトリビュートコードの出力、および比較のために用いる。 |
| BCレジスタペア | : VRAMのアドレス指定に用いる。 |

フローチャートを見ると、まだ何か足りませんね。そう、ループ回数のカウントです。これには、A, B, C以外なら構いませんが、ここでは、Dレジスタをあてましょうか。

| | |
|-------|-------------------|
| Dレジスタ | : ループ回数のカウントに用いる。 |
|-------|-------------------|

そして、フローチャートにある「回数を1減ずる」を実現するために、次のマシン語を覚えましょう。

解 説

ニーモニック: DEC D

マシンコード: 15

機能: Dレジスタの内容を1減ずる。フラグ変化は、
●Cyフラグは変化しない。
●Zフラグは、結果が0なら1に、結果が0以外なら0に変わる。

[注] Cyフラグが不変というのは解せないかもしれませんが、「1だけ減ずる」という特別な減算命令だからだと、ここでは納得しておいて下さい。DEC は、decrement の略。

回数のカウントができれば次はループさせるためのジャンプ命令がなくてはなりません。DEC D によりZフラグが変化することを利用し、次の条件ジャンプ命令を用います。

解 説

ニーモニック： JP NZ, nn'

[(注) nn' はアドレスを指定する
16ビット数値(16進4桁)]

マシンコード： C2 n' n

機 能： ゼロフラグが0ならば、nn'番地へ
ジャンプする。そうでなければ何もしない。

ジャンプ先のアドレスを示す2バイト数値 nn' をアセンブルする時は、「上下位逆転の原則」を適用するのでしたね。

以上で私たちは当面必要とするマシン語をすべて手に入れたことになります。さっそくプログラムを書き上げましょう。例によってプログラムはメモリー上 D000H番地から配置いたします。

図3-12 ループを用いた♥4個表示プログラム

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|--------------|----------------|
| D000 | 01F431 | LD BC,31F4H | …VRAM初期アドレス |
| D003 | 1604 | LD D,04H | …ループ回数4 |
| D005 | 3EE3 | LD A,0E3H | ♥を出力 |
| D007 | ED79 | OUT (C),A | |
| D009 | CBA0 | RES 4,B | …アドレスVRAMへ移行 |
| D00B | 3E07 | LD A,07H | アドレス出力 |
| D00D | ED79 | OUT (C),A | |
| D00F | CBE0 | SET 4,B | …テキストVRAMへ復帰 |
| D011 | 03 | INC BC | …VRAMアドレスを1進める |
| D012 | 15 | DEC D | …回数を1減ずる |
| D013 | C205D0 | JP NZ,0D005H | …条件ジャンプ |
| D016 | C9 | RET | …プログラムの停止 |

条件ジャンプの所、少し説明を加えておきます。DEC D により、Dレジスタの内容は初期値4から1ずつ減ってゆきますが、0にならぬうちは、Zフラグは立ちませんから、

J P N Z, 0 D 0 0 5 H によりCPUの制御は D 0 0 5 H番地へ移ります。こうしてループができるのですが、4回目に D E C D を実行すると、Dレジスタの内容が0になり、Zフラグが立つので、条件ジャンプはそのまま通過して、D 0 1 6 H番地の R E T を実行し、モニターのコマンドレベルへ戻ってプログラムは停止します。

プログラムの理解ができましたら、モニターを起動し、マシンコードを入力して下さい。
できたら * G D 0 0 0 で実行です。画面中央から右に ♥ が4個表示されましたか？

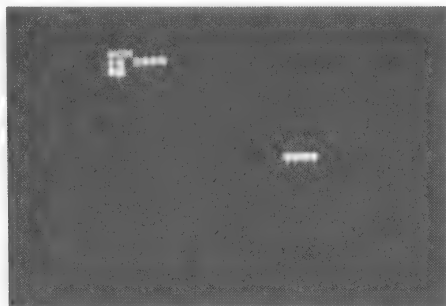


図3 - 1 3

プログラム中、条件ジャンプ命令 J P N Z, n n' が登場しましたが、よい機会ですから、ここでZフラグとC yフラグによる条件の表記法をまとめておきます。

Z「ゼロフラグが1であれば」
N Z「ゼロフラグが0であれば」
C「キャリーフラグが1であれば」
N C「キャリーフラグが0であれば」

キャリーフラグの条件記号の C とレジスタの C を混同しないように注意して下さい。
付録の「Z 8 0 命令表」にある条件ジャンプ命令

J P Z, n n'
J P C, n n'
J P N C, n n'

の意味も、もうおわかりですね。

以上で、私たちは、マシン語による条件判断（フラグの使い方）を学び、BASICでの
FOR ～ NEXT にあたるループ処理をマスターしたことになります。ループ回数を変えたり（この方法で255回までループできます）、アトリビュートや表示位置を変えたりして、いろいろ実験してみてください。

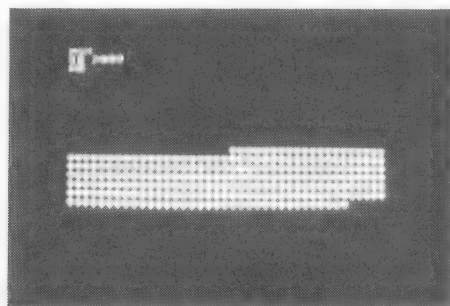


図3-14

☞ ティー・ブレイク

一息いれましょう。コーヒーばかりでは体によくなさそうなので、今回は紅茶にしました。ここまでの道のり、いかがでしたか？ フラグというコンピューター独特の概念を理解するまで、私も随分分かりました。プロの人たちが作ったプログラムを読むと、フラグを魔術のように用いて、省メモリー・高速化を実現しています。私には、まだそのような芸術的（ともいえる）プログラムは書けませんが、一步一步着実に前進してゆきましょう。

フラグと並んで、（私を含め）初心者にとって、難関となる概念に、スタックがあります。後で登場してきますから、その時は、心して取り組んで下さい。

3-12／画面反転プログラム

マシン語によるループ処理のまとめとして、1画面の色を反転するプログラムを作成しましょう。

まず、画面反転とは、どのようなものかを経験するために、テンキーの - キーを、CTRL を押しながら押して下さい。次に CLR キーで、画面を消去してみてください。いかがですか？ 画面が真白になりましたか？

これが1画面の反転です。もとに戻すには、再び CTRL + テンキー - を押して、CLR して下さい。

X1では、どのようにして画面反転を実行しているのでしょうか。

3-4節で、アトリビュートの勉強をしたとき、1バイトのアトリビュート情報の第3ビットが反転モードを意味すると述べました。

つまり、画面のある位置のアトリビュートを反転モードにするには、対応するアトリビュートVRAMの第3ビットを1にすればよいのです。これを、画面の左上隅から、右下隅まで全域にわたって繰り返せばよいことになります。1画面(40×25モード)は、1000文字分ありますから、1000回のループ処理を行なう必要があります。

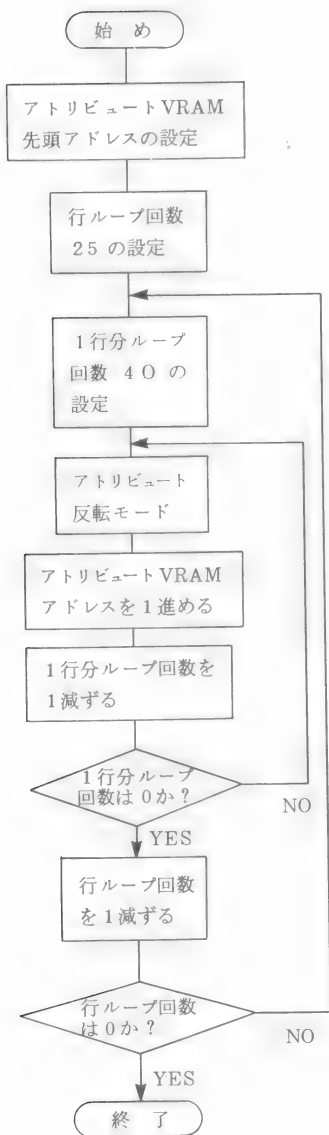
前節でマスターしたのは、255回までのループ処理ですから、当然足りません。では、どうすればよいか？少なくとも2つの方法が考えられます。

方法1： 前節のループ処理を画面1行分(40字)で使い、もう1つ行のループを考える2重ループ法。

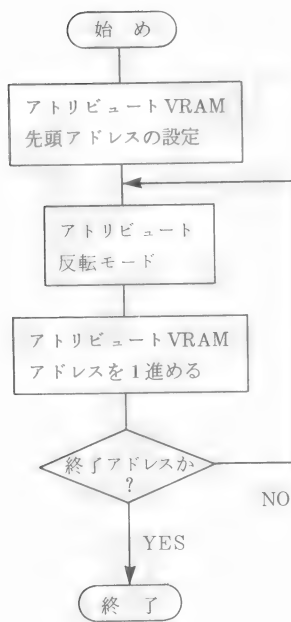
方法2： あらかじめ、終了アドレスを設定しておき、一回ごとに比較を行なう2バイトのアドレス比較法。

図3-15 《フローチャート》

方法 1



方法 2



これら2つの考え方は、いずれも大切ですから、2通りの方法で画面反転をしましょう。

いずれの方法でも、アトリビュートVRAMのアドレス指定は、BCレジスタペアで行ないます。X1では、そのハードウェア上の特徴（3-6節参照）から、画面表示のために、OUT (C), A 命令を多用するので、BCレジスタペアを頻繁に使います。

画面のある位置のアトリビュートを反転モードにするには、その位置のアトリビュートデータを読み、その第3ビットを1にして、再び出力するという手続きをとればよいのです。BCレジスタペアで指定されるアトリビュートVRAMのデータを読むには、入力命令 IN A, (C) を用います。その第3ビットを1にするには、SET 3, A とすればよく、かくして、Aレジスタにできあがった新しいアトリビュートデータを、OUT (C), A で、アトリビュートVRAMへ出力します。VRAMアドレスを1つ進めるには、INC BC を用いるのでしたね。

さて、方法1のフローチャートのマシン語化に取り組みましょう。2つのループの回数をカウントしなければなりません。BASICでも、FOR ~ NEXT の2重ループでは、外側のループ変数と、内側のループ変数をきちんと区別しなくてはなりません。マシン語はBASICのように、変数が使えませんが、かわりに、レジスタを用いるのです。今の場合、D, Eレジスタは他に用いせんから、外側のループ（25行分）をDレジスタで、内側のループ（1行40字分）をEレジスタで、カウントすることにしましょう。

図3-16 《2重ループ（方法1）による画面反転プログラム》

| アドレス | マシンコード | ニーモニック | コメント | |
|------|--------|--------------|--|--|
| D000 | 010020 | LD BC,2000H | アトリビュートVRAM先頭アドレス設定 | |
| D003 | 1619 | LD D,19H | <div style="display: inline-block; vertical-align: middle; text-align: center;"> <div style="border-left: 1px dashed black; border-right: 1px dashed black; height: 100px; margin: 0 auto; width: 20px;"></div> <div style="display: flex; flex-direction: column; align-items: center; justify-content: space-between; padding: 0 5px;"> 内側ループ 回数40 (=28H) </div> </div> <div style="display: inline-block; vertical-align: middle; text-align: center; margin-left: 20px;"> <div style="border-top: 1px dashed black; border-bottom: 1px dashed black; height: 100px; margin: 0 auto; width: 20px;"></div> <div style="display: flex; flex-direction: column; align-items: center; justify-content: space-between; padding: 0 5px;"> 外側ループ 回数25 (=19H) </div> </div> | |
| D005 | 1E28 | LD E,28H | | |
| D007 | ED78 | IN A,(C) | | |
| D009 | CBDF | SET 3,A | | |
| D00B | ED79 | OUT (C),A | | |
| D00D | 03 | INC BC | | |
| D00E | 1D | DEC E | | |
| D00F | C207D0 | JP NZ,0D007H | | |
| D012 | 15 | DEC D | | |
| D013 | C205D0 | JP NZ,0D005H | | |
| D016 | C9 | RET | プログラムの停止 | |

上が完成したプログラムです。メモリーの D000H番地から格納しました。2重ループの部分、よろしいですか？ モニターから入力して、実行して確認してみてください。

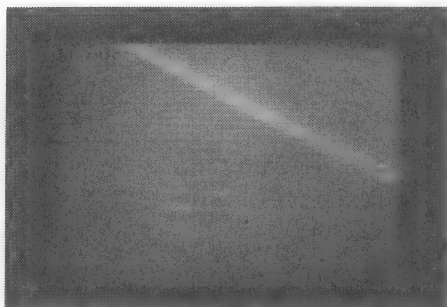
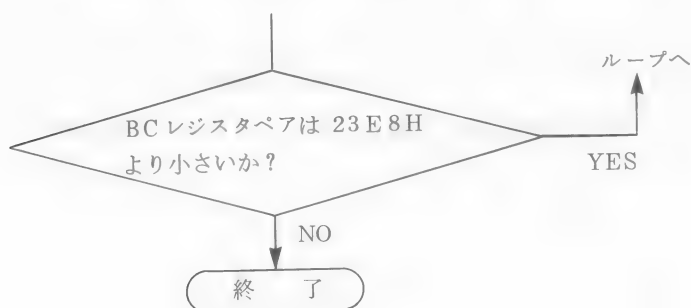


図3-17

では次に、方法2を考えます。この方法は、ループとしては一重、しかし、処理終了アドレスとBCレジスタペアを比較（2バイト比較）しなければなりません。（1ページ目の）画面右下隅に対応するアトリビュートVRAMのアドレスは、23E7H です。このアドレスのアトリビュートを反転モードにした後、INC BC により、アドレスは1つ進むはずですから、ループを抜け出す時には、BCレジスタペアは、23E8H を示しているはずです。そこで、次のような条件判断を行なってみましょう。



付録の「Z80命令表」の比較命令の中には、私たちが必要とする2バイト（16ビット）比較命令はありませんね。では、どうすればよいか？

答は意外に簡単です。1バイトずつ2段階に分けて比較すればよいのです。ただし、上位バイトがBレジスタ、下位バイトがCレジスタであることに注意して、Bレジスタと23Hの比較を先に行なわねばなりませんね。また、Bレジスタと数値を直接比較する命令也没有から、まずAレジスタにBレジスタの内容を取り込まねばなりません。「小さければ」の条件判断はCyフラグで行なうのでした。

以上により、次のプログラムができました。

図3-18 《アドレス比較（方法2）による画面反転プログラム》

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|-------------|------------|
| D000 | 010020 | LD BC,2000H | |
| D003 | ED78 | IN A,(C) | |
| D005 | CBDF | SET 3,A | |
| D007 | ED79 | OUT (C),A | |
| D009 | 03 | INC BC | |
| D00A | 78 | LD A,B | |
| D00B | FE23 | CP 23H | …Bを23Hと比較 |
| D00D | DA03D0 | JP C,0D003H | …小さければループへ |
| D010 | 79 | LD A,C | |
| D011 | FEE8 | CP 0E8H | …CをE8Hと比較 |
| D013 | DA03D0 | JP C,0D003H | …小さければループへ |
| D016 | C9 | RET | …プログラムの停止 |

例によって、入力、実行、確認をして下さい。予想どおりでしたか？

これらの画面反転プログラムは、例えば、ゲームプログラムでの、「GAME OVER 画面」などに応用すると面白いでしょう。私たちは、次章において、マシン語ゲームの作成に取り組む予定ですが、そこに画面反転を応用してみるつもりです。楽しみにして下さい。

【応用問題】

40×25 モードで、1画面消去のプログラムを作ってください。

ヒント： 文字を消すというのは、その位置に、
空白（スペース）のASCIIコード 20H
を出力することと同じです。

3-13／問題点の整理

前節において私たちは、画面反転のプログラムを完成いたしました。この種のプログラムは単独で用いるというよりは、ゲームプログラム等もっと大きなプログラムの中に組み込んで用いるという性質のものです。

私たちは、画面反転プログラムを仮にメモリーの D000H番地から配置しテストしましたが、上のことを前提に考えると「メモリーのどこに配置するか？」について、もう少し気を配る必要がありそうです。すなわち、画面反転プログラムを組み込むべき本体プログラムが決まらない以上、私たちはメモリーのどの番地に配置するか決定できないことになります。

これを裏返して言うと、どのような本体プログラムに組み込まれても、メモリーのどの番地に配置されても、正常な動作をすることが要請される訳ですね。このような視点で、私たちが作成した画面反転プログラムを反省してみましょう。方法1、方法2いずれの場合も、ループ処理のために条件ジャンプ命令（JP NZ, nn' または JP C, nn'）を用いていますが、これらはジャンプ先のアドレスをきちんと指定してしまっているのです。これらのプログラムは D000H番地から配置された時にのみ正常な動作をします。すなわち、私たちのプログラムでは上の要請に応えることはできない訳です。

メモリー上の配置番地を変更しても、正常な動作をするプログラムを、リロケートブル（relocatable=再配置可能）であるといいます。この用語を用いると、前節での画面反転プログラムはリロケートブルではない！ といえます。すなわち、本節で提起される課題は次のものです。

課題 リロケートブルな画面反転プログラムを作ること。

3-14/リロケータブルとは？

本節では、アドレス比較（方法2）による画面反転プログラムを材料に、リロケータブルの問題を考えてゆきましょう。前節で指摘した通り、このプログラムをリロケータブルでなくしている原因は、2か所で使われている条件ジャンプ命令 JP C, 0D003H であることは明らかです。

もう少し具体的に説明してみましょう。条件ジャンプ命令 JP C, 0D003H が実行されると、条件成立時には常に D003H番地にジャンプしてしまいます。ではもし、この画面反転プログラムを E000H番地から配置して実行した時、ジャンプ先の D003H番地に「未知のマシン語」が書かれていたら、どうなるでしょう。そうです、何が起きるかわかりませんね。最悪の場合は「暴走」を起こすことでしょう！

画面反転プログラムをたとえば E000H番地から配置しても、正常な動作をさせるためには、ジャンプ先のアドレスを D003H番地に固定してはならないのです。そのためにはどうすればよいのでしょうか？

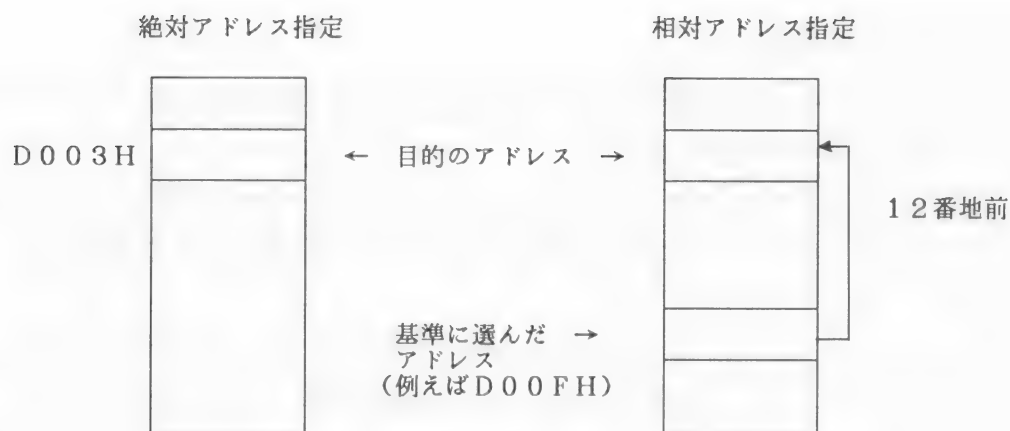
私たちは、ある場所を指定するのに、どうしているでしょう。たとえば、住所なら、〇〇県 〇〇市〇〇町〇丁目〇番地〇号 などと指定しますね。郵便物は、こうして配達される訳です。

しかし、これだけでしょうか。旅行に出かけて、ある名所旧跡を訪れたいが、よくわからないとします。ガイドブックなどで、その近所までは何とか行けました。さて次は？

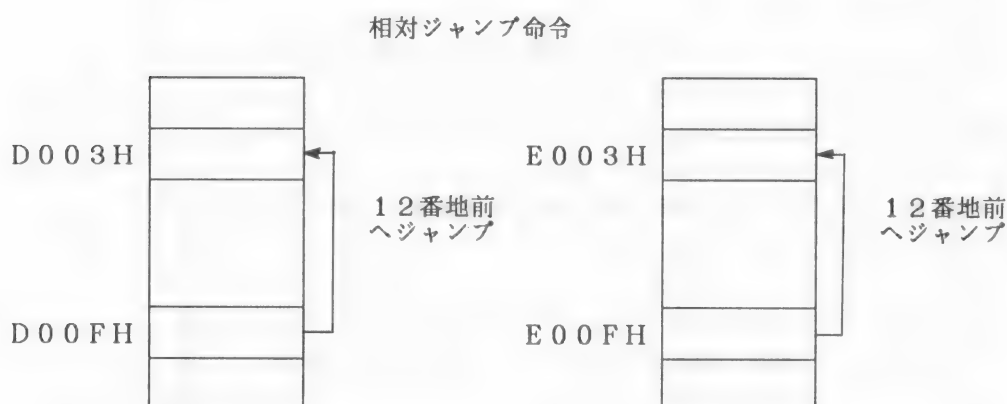
私たちは、土地の人たちや交番で尋ねます。「〇〇へ行きたいのですが、どうすればよいでしょう？」 すると、次のような答が大抵返ってくるはずです。「〇〇は、ここからその角を左へ曲がって、〇〇m位行けばありますよ。」

マシン語においても、このような形式のアドレス指定があります。すなわち、あるアドレスを基準にして、（相対的に）目的のアドレスが、何番地後にあるのか、前にあるのか、という指定法です。これを、相対アドレス指定とよびます。これに対して、今まで使ってきた 〇〇番地 式のアドレス指定は絶対アドレス指定とよばれます。

2つのアドレス指定の違いを図にすると次のようになります。



私たちが、今、リロケートブルの問題と関係して必要としているのは、相対アドレス指定によるジャンプ命令 —— 相対ジャンプ命令 という —— なのです。



上図により、相対ジャンプの形式でプログラムが書ければ、格納アドレスを変更しても、同じ結果が得られ、リロケートブルとなるのがわかります。

付録の「Z80命令表」のジャンプ命令の項を見ると、JR … という形の命令がありますね。これらが相対ジャンプ命令です。(ニーモニックで、JR の部分は、Jump Relative の略です)

では、相対アドレスによりジャンプ先を指定するには、どうするのでしょうか？ 特に、どこを基準アドレスに採るのでしょうか？

3-15/相対ジャンプ命令

第2章で、Z80の全レジスタを紹介した時、PC（プログラム・カウンタ）という16ビットレジスタがあったことを思い出して下さい。

このレジスタは、CPUが今、メモリーのどのアドレスに注目しているかを記憶しておくために設けられています。通常、CPUが命令を実行するごとに、PCの値は1ずつ自動的に増加されます。

では、PCに、（16ビットの）全く別の数値を入れたらどうなるでしょう。CPUは、その数値を次に実行すべきアドレスと認めて、そのアドレスに制御を移してしまいます。すなわち、ジャンプします。そうです！ JP C, nn' という命令は、Cyフラグが立った時に、PCに値 nn' をロードする命令と考えることができる訳です。

$$\text{JP C, nn'} = \begin{cases} \text{Cyフラグが立った時は、} & \text{PC} \leftarrow \text{nn'} \\ \text{Cyフラグが立たない時は、} & \text{そのまま。} \end{cases}$$

さて、相対ジャンプに話を戻しましょう。次の命令を採り上げます。

解 説

ニーモニック： JR e

マシンコード： 18 e [eは1バイト数値]

機能： 相対アドレス e で指定された番地へ
ジャンプする。

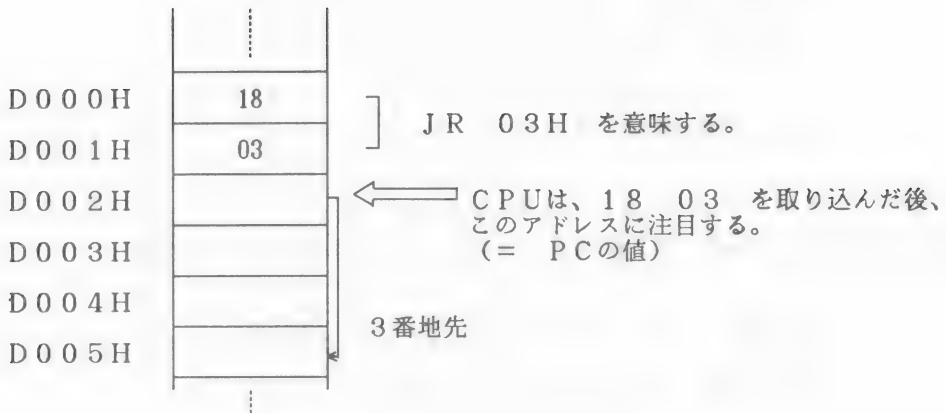
相対アドレス e の説明をいたします。

まず、基準にするアドレスですが、それは、

CPUが、JR e という命令を取り込んだ直後の
プログラム・カウンタの内容

です。もう少しかみくだきましょう。例えば、メモリーに次のように格納されていたとします。

《相対ジャンプのしくみ》



CPUは、命令を次々と実行して、D000H、D001H番地に格納されている2バイト命令 JR 03H を取り込みました。すると自動的にプログラム・カウンタは1増えますから、この瞬間のPCは、D002Hを示しています。さて、JR 03Hはどういう命令かという、PCの値に、03Hを加えてしまうものなのです。すると、どうなりますか。PCの値は、D002H+03H=D005Hとなりますね。そこで、CPUは、3番地後のD005H番地へ制御を移します。つまり、D005H番地へジャンプします。おわかりですか？ 一般に、

$$JR\ e = PC \leftarrow PC + e$$

となって、JR e とはPCに関する加(減)算命令に他ならないのです！

では、手前にジャンプするには、どうすればよいのでしょうか？ e が負の数であればよいですね。

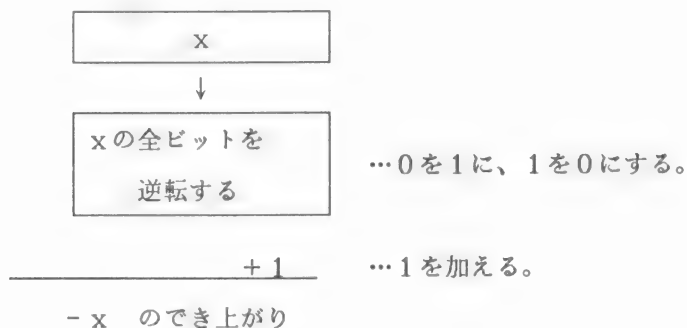
8ビットで、負数を表すには、2の補数という考え方をします。たとえば、-1を8ビットで表すには、どうするかというと、

$$\begin{array}{rcl}
 \text{2進数} & \left\{ \begin{array}{l} 00000001 = 01\text{H} \\ + 11111111 = \text{FFH} \end{array} \right. & \\
 \text{でたし算} & \begin{array}{r} \hline 1 \\ 00000000 \end{array} & \\
 & \begin{array}{l} \uparrow \\ \text{無視} \end{array} &
 \end{array}$$

を $1 + (-1) = 0$ と見なして FFH により表わします。同様に

$$\begin{array}{rcl}
 \text{2進数} & \left\{ \begin{array}{l} 00000010 = 02\text{H} \\ + 11111110 = \text{FEH} \end{array} \right. & \\
 \text{でたし算} & \begin{array}{r} \hline 1 \\ 00000000 \end{array} & \\
 & \begin{array}{l} \uparrow \\ \text{無視} \end{array} &
 \end{array}$$

ですから、 -2 は FEH で表わします。これらの作り方を反省すると、一般に、 x を8ビットの数とすると、 $-x$ を作るには、

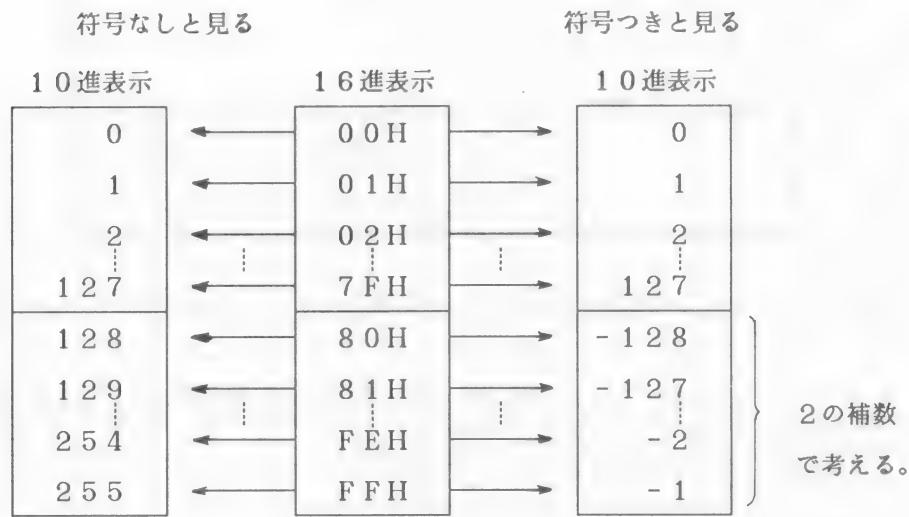


となります。

さて、8ビットで正負を区別するには、その最上位ビット（MSB）により行ないます。すなわち、符号付数と見なすには、

$$\begin{array}{lcl}
 \text{MSB} & = & 0 \Rightarrow \text{正（または0）の数} \\
 \text{MSB} & = & 1 \Rightarrow \text{負の数}
 \end{array}$$

と決めています。こうして次の2通りの見方ができることになります。



話を戻します。相対ジャンプ JR e を考えていました。ここで、相対アドレスを表わす1バイト数値 e は、符号付の数と考えるのです(10進数では -128~+127)。すなわち、3番地手前にジャンプするには、 -3 を2の補数で考えて、 e = FDH とすればよく、

JR 0FDH = 3番地手前にジャンプ

となります。付録2に、「1バイト符号付16進数」の表がありますから、確認し、よく練習しておいて下さい。

3-16／画面反転をリロケータブルに！

かくして、私たちは、画面反転プログラム(方法2で考えましょう)をリロケータブルにする方法をマスターしました。2つの条件ジャンプ命令を、相対ジャンプで置きかえてみましょう。

解 説

ニーモニック： JR C, e

マシンコード： 38 e

機能： Cyフラグが1ならば、相対アドレス
e ヘジャンプする($PC \leftarrow PC + e$)。
Cyフラグが0ならジャンプしない。

必要なのは、上のような条件付相対ジャンプ命令ですね。完成したプログラムを掲げます。

図3-19 《相対ジャンプを用いた画面反転プログラム》

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|-------------|--------------------|
| D000 | 010020 | LD BC,2000H | |
| D003 | ED78 | IN A,(C) | |
| D005 | CBDF | SET 3,A | |
| D007 | ED79 | OUT (C),A | |
| D009 | 03 | INC BC | |
| D00A | 78 | LD A,B | |
| D00B | FE23 | CP 23H | Bが23Hより小さければ12番地前 |
| D00D | 38F4 | JR C,0F4H | … ヘジャンプ (PC=D00FH) |
| D00F | 79 | LD A,C | |
| D010 | FE E8 | CP 0E8H | CがE8Hより小さければ17番地前 |
| D012 | 38EF | JR C,0EFH | … ヘジャンプ (PC=D014H) |
| D014 | C9 | RET | … プログラムの停止 |

ニーモニックは、人間にとってわかりやすく記述するためにあります。上のプログラムのニーモニック表記では、相対アドレスをそのまま書いてありますが、これではどこにジャンプするのか見にくいですね。そこで、ニーモニック表記においては、相対ジャンプであっても相対アドレス値 e のかわりに、実際のジャンプ先を記すことがよく行なわれます（現実の多くのアセンブラでも受けつけてくれます）。以下に掲げるのはこの形で書いたリストです。

図3-20

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|---------|-------------|-------------|-----------|
| D 0 0 0 | 0 1 0 0 2 0 | LD BC,2000H | |
| D 0 0 3 | E D 7 8 | IN A,(C) | |
| D 0 0 5 | C B D F | SET 3,A | |
| D 0 0 7 | E D 7 9 | OUT (C),A | |
| D 0 0 9 | 0 3 | INC BC | |
| D 0 0 A | 7 8 | LD A,B | |
| D 0 0 B | F E 2 3 | CP 23H | |
| D 0 0 D | 3 8 F 4 | JR C,0D003H | …実際のジャンプ先 |
| D 0 0 F | 7 9 | LD A,C | |
| D 0 1 0 | F E E 8 | CP 0E8H | |
| D 0 1 2 | 3 8 E F | JR C,0D003H | …実際のジャンプ先 |
| D 0 1 4 | C 9 | RET | …プログラムの停止 |

実際にリロケータブルであることを確認してみます。そのために、このプログラムをたとえば E 0 0 0 H 番地へ転送してみましょう。

このような操作に便利なモニターコマンドがあります。

モニター T コマンド

《書式》 *T 先頭アドレス 最終アドレス 転送先頭アドレス

《機能》 指定した範囲（先頭アドレスから最終アドレスまで）のデータを転送先頭アドレス以後に転送する。

ではモニターを起動し、次のように入力実行して下さい。

```

MON
*T D000 D014 E000
*

```

図3-21

期待通り転送されているか確認してみます。予想だと、E000H番地からE014H番地までに転送されているはずですから、*D E000 E014 によりダンプします。

《*D E000 E014 による転送の確認》

```

:E000=01 00 20 ED 78 CB DF ED /.. xヒ°
:E008=79 03 78 FE 23 38 F4 79 /y.x生#8ホy
:E010=FE E8 38 EF C9 00 00 00 /生X8□ノ...

```

いかがですか？ 予想通り転送されていますね。E000H番地から格納された画面反転プログラムについて、ニーモニック表記をすると次のようなリストになります。

図3-22 《E000H番地から格納》

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|-------------|-----------|
| E000 | 010020 | LD BC,2000H | |
| E003 | ED78 | IN A,(C) | |
| E005 | CBDF | SET 3,A | |
| E007 | ED79 | OUT (C),A | |
| E009 | 03 | INC BC | |
| E00A | 78 | LD A,B | |
| E00B | FE23 | CP 23H | |
| E00D | 38F4 | JR C,0E003H | …実際のジャンプ先 |
| E00F | 79 | LD A,C | |
| E010 | FEE8 | CP 0E8H | |
| E012 | 38EF | JR C,0E003H | …実際のジャンプ先 |
| E014 | C9 | RET | …プログラムの停止 |

相対アドレスから実際のジャンプ先を計算すると、E003H番地になりますね。つまり、このプログラムは、E000H番地から配置しても、D000H番地から配置したのと同じ働きをすることになります。こうして、私たちは画面反転プログラムをリロケートブル化することに成功しました！

3-17／サブルーチンとスタック

画面反転プログラムの完成をめざしてきた私たちの試みは前節において、一応満足できる段階に達したことになります。そこで本節以降は、このプログラムに関して唯一不透明になっている部分 —— 第1章以来の宿題 —— である「プログラム停止のためのおまじない C9」に関して説明していくことにいたします。

まず、私たちがすでに慣れ親しんでいるモニターのGコマンドについて、マニュアルの正式の説明を読むことにいたします。マニュアル176ページを御覧ください。

***G** ゴーサブ

***G** コールアドレス

指定したコールアドレスをサブルーチンコールします。

スタックポインターは、FFFE（16進）にあります。

とありますが、おわかりですか？ この記述だけでわかる方は、マシン語について基礎知識のある方と思われますので、本節以降の本章の内容は読みとばして構いません。第4章へ進んで下さい。本章では、マシン語に初めて接した読者の方々を対象に、上のマニュアルの記述の完全理解をめざすことにいたします。

まずGコマンドの名称が単に「ゴー」ではなく「ゴーサブ」となっていることに注意しましょう。このことは、Gコマンドによるマシン語プログラムの実行のされ方が、BASICで言う所のGOSUBによる実行のされ方と同様であることを暗示しています。BASICではGOSUB文は、サブルーチンの呼び出しのために用いられますね。ということは、Gコマンドが「マシン語サブルーチン」の呼び出し（コールという）を行なうコマンドであることを意味していますね。こうして本節以降の私たちの目標は次のようになります。

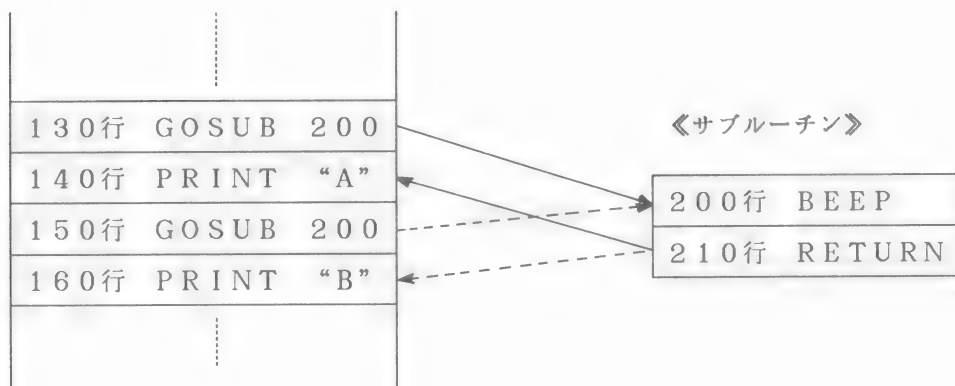
目標 マシン語サブルーチンとその実行の原理について
理解すること。

BASICにおけるサブルーチンとその実行のされ方について復習しましょう。次に簡単なBASICプログラムを掲げます。

```
100 REM —— BASIC ノ サブルーチン ——  
110 /  
120 CLS  
130 GOSUB 200  
140 PRINT "A"  
150 GOSUB 200  
160 PRINT "B"  
170 END  
180 /  
190 /  
200 BEEP  
210 RETURN
```

行番号170までがメインルーチン、行番号200以降がサブルーチンですね。実行のされ方は次図のようになります。

《メインルーチン》



BASICインタプリタは、GOSUB文に出会うと戻り行をメモリー上のどこかに保存し、それから GOSUB文の示すサブルーチンへジャンプしてゆきます。さて、サブルーチンの末尾へ来て、RETURN に出会うと保存しておいた戻り行を復帰させ制御をそこに移す —— 以上が、GOSUB によるサブルーチン実行の原理です。

BASICプログラムにおいては、BASICインタプリタがすべてを管理してくれますが、私たち自身が作成するマシン語プログラムでは、BASICインタプリタの手を借りずに直接CPUに働きかける点が異なっています。しかし、マシン語においてもサブルーチンへのジャンプとメインルーチンへの復帰の原理は全く同様です。

CPUが、次々とマシン語命令を実行していく過程で、サブルーチンからの戻り番地のように保存しておくべきデータがいくつか生じてきます。データの一時的保管場所であるレジスタでは数に限りがありますから、このような場合は、メモリーの一部を保管場所として割りあてるということをいたします。この目的に用いられるメモリー上の領域をスタックとよびます。スタック (stack) を辞書で引くと、「干し草の山、積み重ね」と出ています。コンピュータの場合、「積み重ね」られるものは（保存しておきたい）データですが、次のようなイメージでとらえるとわかりやすいでしょう。

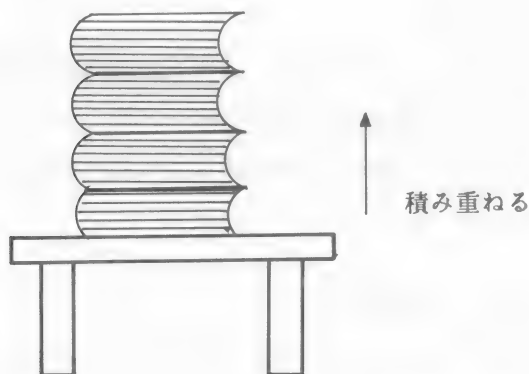


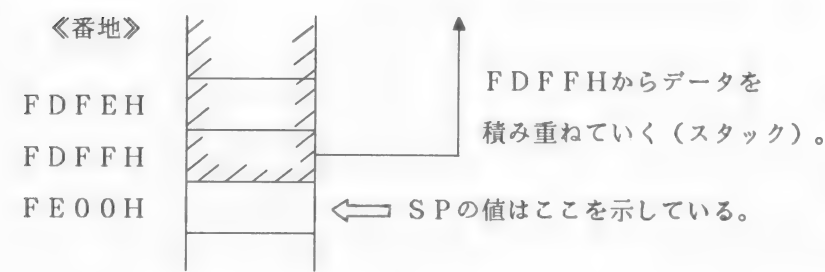
図3-23

上の絵の場合は本を積み重ねていますが（いわゆるツン読）、積み重ねるためには当然のことですが、机などの台がなくてはいけませんね。コンピュータでも同様で、データをどこから積み重ねていけばよいのか —— スタックがどこなのか？ —— をCPUは知っていません。そのために設けられているレジスタがスタックポインタ (SP)です。

3-18/スタックポイントの働き

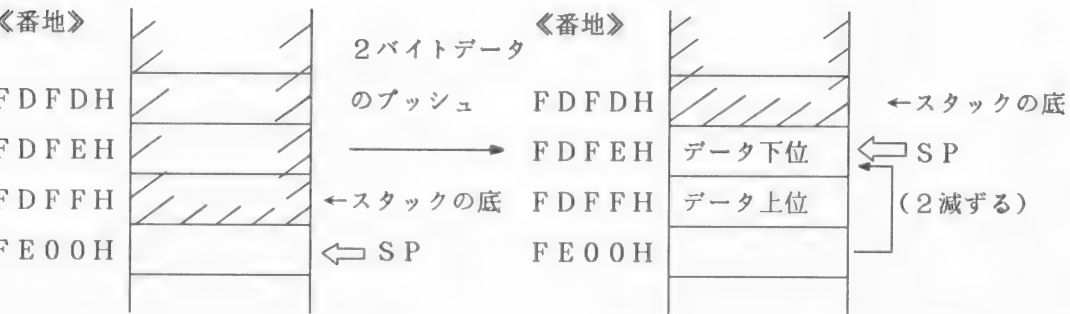
スタックポインタ（記号SP）は、16ビットのレジスタで、スタックの位置を記憶しておくために用いられます。

たとえば、SPに16ビット数値 FE00H がセットされていたとします。このときスタックは次のように設定されていることを意味します。



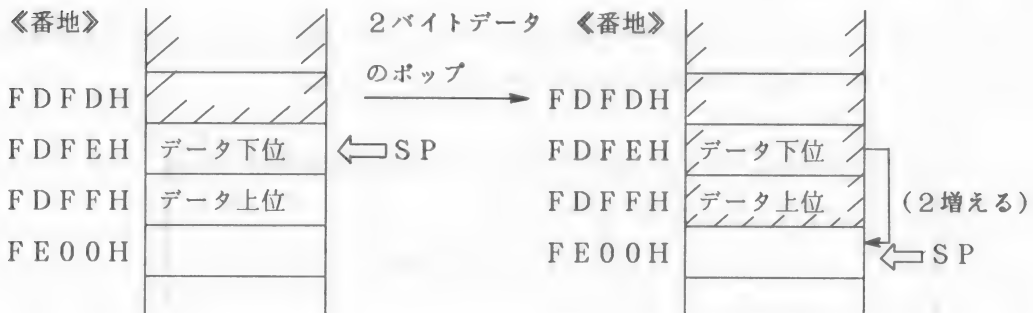
図の場合、スタックはFDFFH番地から番地の数字の若い方に向かって設けられています。一般に、SPの示す値から1を引いた値が、スタックの一番“底”を示しています。データはこの番地から若い方に積み重ねられるのです。

では、スタックへのデータの積み重ね（プッシュ = push という）について説明いたします。まず大切なことは、データは2バイトずつ積み重ねられる点です。従って、スタックへのデータのプッシュが行なわれると、SPの値は自動的に2減ずるようにCPUは設計されています。



上図を見ていただくと、SPの働きがわかると思いますが、2バイトデータの積み重ねにおいても「上下位逆転」が生じることに注意して下さい。

逆に、CPUがスタックに積んでおいたデータを取り込む（ポップ=pop という）ときはどうでしょう。SPの値は、積み重ねられたデータの一番「上」を示しているはずですから、CPUはSPの値を参照して、そこから2バイト分のデータを取り込めばよいのです。同時に、SPの値は自動的に2増加するようになっています。



CPUが2バイトデータをポップした後も、積み重ねたデータはメモリーに残りますが、もはやCPUは関知しません。そこはスタックの底と見なされますから、次に新しいデータをプッシュする時は、平気で書き換えてしまいます。

CPUとスタックとのデータのやりとり（プッシュとポップ）、それに伴うSPの動きは理解できましたか？ このことがわかると次の重要な点に気づくはずです。

スタックに積み重ねられているデータは大切なものであるから、これを理由なく破壊してはならない。そのためにも、SPがどこを示しているか把握しておく必要がある。

私たちは、今まで気楽にモニターを起動しマシン語プログラムを入力・実行してきました。幸い今まではトラブルなく済んでいたことでしょう。しかし、もし偶然に、プログラムを格納する場所が、BASICインタプリタやモニターの使用するスタックと重なってしまったら

どうなるでしょう。スタックに保存されているデータは、それらのシステムプログラムにとって大切なデータであるはずですから、これを勝手に変更してしまうと、最悪の場合、BASICやモニターに戻った時、システムが壊れて「暴走」するかもしれませんね。

このような危険を避けるためにも、私たちはBASICインタプリターやモニターが動いている時のSPの値について知識を持たねばなりません。

3-19／システムの使用するスタック

16ビットレジスタであるSPに値を設定するためのマシン語があります。「Z80命令表」の16ビットロード命令の項を見ると次の命令が見つかります。

解 説

```
ニーモニック： LD  SP, nn'  
                (nn' は2バイト数値)  
マシンコード： 31  n' n  
機      能：   SP ← nn'
```

X1の電源をONし、BASICのシステムテープのロードが終了すると、CPUはまずシステム初期化ルーチン(00FAH番地から始まっている)を実行します。このルーチン内では、

LD SP, 0000H

が実行されて、最初のスタックは FFFFH番地より若い方に向かって設定されます。試みに、モニターを起動し、 *D FFC0 FFFF によりメモリー内容をダンプして下さい。


```

:FDC0=00 00 00 00 00 00 00 00 /.
:FDC8=00 00 00 00 00 00 00 00 /.
:FDD0=00 00 00 00 00 00 4D 0B /.
:FDD8=CD 21 55 03 08 08 12 08 /^!U.
:FDE0=CA 03 76 A0 26 07 03 00 /h.v &...
:FDE8=C3 04 5B 9E 43 70 5C 9E /テ.[ Cp¥r
:FDF0=54 9D 4C 6F DD 6E 44 FF /T'LoFnD+
:FDF8=64 20 55 9D D3 15 FF FF /d U'モ.テテ
:FE00=00 00 00 00 00 00 00 00 /.

```

図3-25

F D F F H番地から若い方へ向かってデータが積み重ねられている様子がわかりますね。これらのデータは特別な必要のない限り破壊するべきではありません。

次に、モニター起動時のスタックについて考えます。 MON を実行すると、BASICインタプリタは、必要なデータをスタックにプッシュした後、

```
LD SP, 0000H
```

を実行し、ここに2バイトデータ 1 0 0 3 H をプッシュします。このデータは、モニターのコマンド待ち処理をするルーチンの開始アドレスです。先に、 *D F F C 0 F F F F によりメモリーダンプをした際、 F F F E H番地と F F F F H番地に格納されていた



という2バイトデータがこれです。上下位逆転して格納されていますね。従って、モニターコ

マンドレベルでのSPの値は、FFFEH番地を示していることになります。

私たちが、モニターGコマンドを実行する時は、このようなスタックの設定（FFFDH番地から若い方へ向かって）になっている訳ですね。3-17節で紹介した「マニュアル176ページ」の説明中、「スタックポインターは、FFFE（16進）にあります。」という記述は以上のことを背景にしている訳です。よろしいですか？

こうして、システム（BASICインタプリタ及びモニター）が使用するスタックの位置がわかりましたから、次の2つの問題を具体的に扱うことができます。

問題1 マシン語プログラムを安心して格納できる領域はどこか？

問題2 モニターのGコマンドによるマシン語プログラムの実行のされ方はどのようなものか？

次節からこれらの問題を解決していくことにいたします。

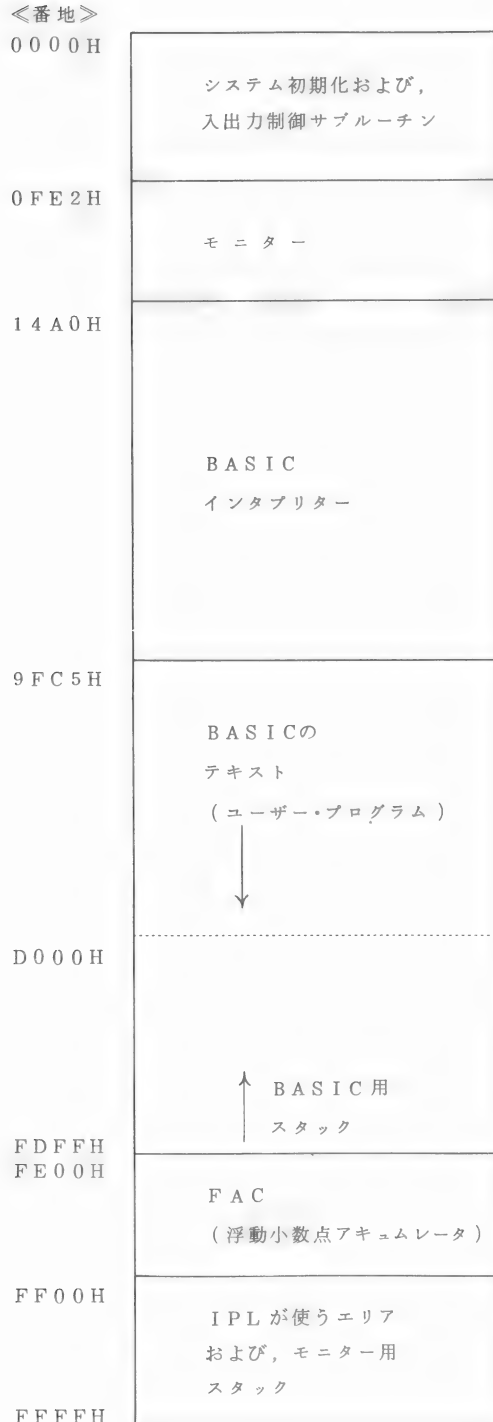
3-20／マシン語フリーエリアの確保

私たちがメモリーに格納するプログラムがシステムプログラムを破壊しないようにするという観点から見ると、私たちが今まで実行してきた操作は偶然にうまくいっていたと言わざるを得ません。

メモリー内のどの場所に何が記憶されているかを示す図をメモリーマップとよんでいます。マニュアル183ページにメモリーマップが出ていますが、これに必要なデータを書き加えてみました。BASICテープをロードした直後のメモリーマップは次のように設定されています。

図3-26

BASIC 起動時のメモリーマップ



このメモリーマップを見ていただくとわかるように、私たちが無意識に使用してきた D000H番地のあたりは、上からはBASICのテキストエリアに迫られ、下からはBASICのスタックに迫られるサンドイッチになった領域であることがわかります。今までは、長いBASICプログラムを組まなかったことと、BASICスタックに多くのデータが積み重ねられなかったという「偶然」により、私たちのマシン語プログラムは正常に走ったのでした。

このような不安定な領域ではなく、もっと確実な領域に私たちユーザーのマシン語プログラムを格納するにはどうしたらよいのでしょうか？ そのために用いられるBASICコマンドが

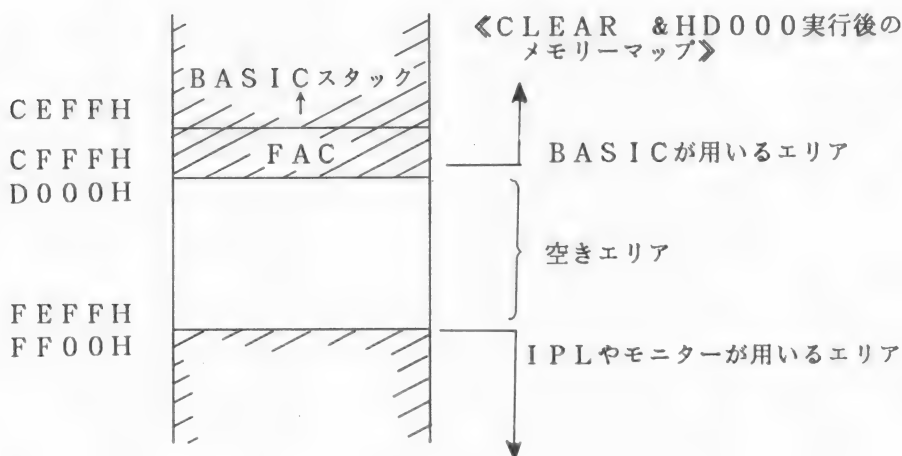
CLEAR

です。X1のBASICでは、このコマンドに同音異義が2つあります。ここで述べるのは、「変数・配列をすべてクリアする」ための CLEAR（マニュアル41ページ）ではなく、マニュアルの20ページにある CLEAR の方です。

このコマンドは次のような書式で用いられます。

CLEAR アドレス（=BASICが使用する上限アドレス+1）

たとえば、CLEAR &HD000 を実行してみましょう。すると、以後BASICインタプリタは、メモリーのCFFFH番地より若い番地の方だけを使用するようになります。



今までBASIC使用エリア内に含まれていた D000H~FEFFH番地に「空き」が生じたことに注目して下さい。この部分こそ、私たちユーザーが安心してマシン語プログラムを格納しておけるフリーエリアなのです。

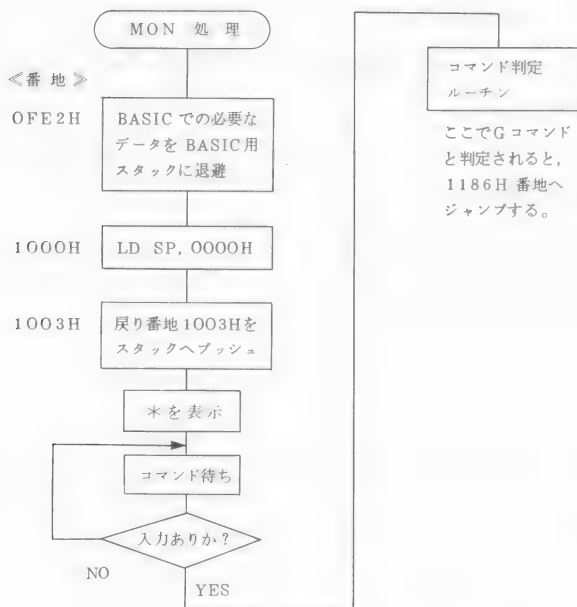
もし、フリーエリアをもっと広くとりたければ、たとえば CLEAR &HC000 を実行すると、C000H~FEFFH番地をフリーエリアとして使えるようになります。ただし、BASIC使用エリアを余り狭くしすぎると、Out of memory エラーが出てしまいます。まあ普通は、CLEAR &HD000 程度にしておくのが無難のようです。

以上のこと、理解されましたか？ 今後はマシン語プログラムをメモリーに格納する際には、積極的に CLEAR を活用して、ユーザー用マシン語フリーエリアを確保するよう努めましょう。こうして前節で提起された問題1は解決いたしました。

3-21/Gコマンドの解明

では最後に残った問題 —— モニターGコマンドによるマシン語プログラムの実行のされ方 —— の解明に取りかかります。

モニターの処理を解読すると、おおよそ次の手順でGコマンドを実行することがわかります。



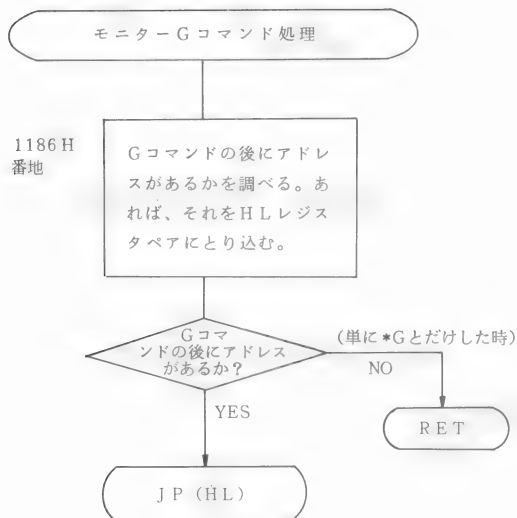


図 3 - 2 7

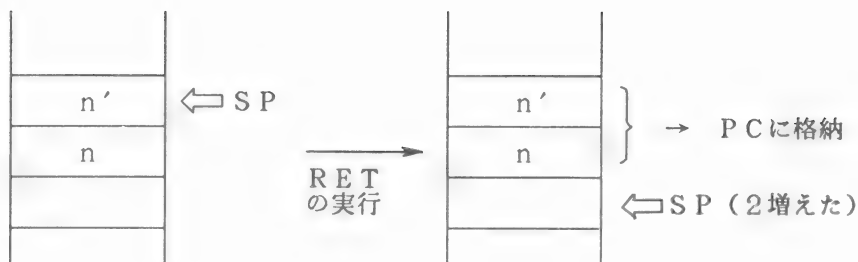
G コマンド処理ルーチンの 2 つの出口にあるマシン語が理解の鍵です。そろそろ、マシン語 RET の正体を述べる時が来たようです。

解 説

ニーモニック： RET

マシンコード： C 9

機能： SP の示す番地、及びその次の番地から
2 バイト分をポップし、プログラムカウンタ
PC に格納する。



モニタールーチンのフローチャートを見ると、Gコマンド実行の時点では、スタックの状況は次のようになっています。

《Gコマンド実行前のスタックの状況》

| 《番地》 | | |
|-------|----|-------------|
| FFFDH | | |
| FFFEH | 03 | ← SPの示すアドレス |
| FFFFH | 10 | |
| 0000H | | |

***G アドレス** を実行すると、1186H番地から始まる「Gコマンド処理ルーチン」へジャンプし、その中で、アドレスの値は2バイトデータとして、HLレジスタペアに格納されます。この場合、「Gコマンド処理ルーチン」の出口には、マシン語 JP (HL) があります。

解説

ニーモニック： JP (HL)

マシンコード： E9

機能： PC ← HL すなわち、HLレジスタペアの示すアドレスへジャンプする。

従って、私たちが今までよくやってきたように、***G D000** を実行すると、「Gコ

マンド処理ルーチン」の出口で、 D 0 0 0 H番地へジャンプすることになります。すなわち、 D 0 0 0 H番地から格納されたマシン語プログラムが実行されます。

さて、私たちは「マシン語プログラムを停止させるおまじない」として、プログラムの最後にマシン語 RET を置いて来ました。なぜこれで、モニターのコマンドレベルに戻るのかは、もうおわかりですね。

プログラム中で、最終的にSPの値を変更しない限り、プログラム末尾の RET の直前に来た時、SPの値は F F F E H番地になっているはずですが。これで RET を実行すると、

《プログラム末尾における RET の実行》



となって、プログラムカウンタPCに 1 0 0 3 H がロードされ、従って、1 0 0 3 H番地から始まるモニターのコマンド待ちルーチンへジャンプできるのです。

3-22／マシン語サブルーチンの実行原理

前節で明らかになったモニターGコマンドの原理は、実は、サブルーチンの実行原理そのものなのです。私たちが作成し、RET により停止させるマシン語プログラムは、モニタープログラムより呼び出される（コールされる）サブルーチンと見なされていることになります。従って、Gコマンドの名前が「ゴースブ」なのですね。

本節では、私たちのマシン語プログラム中から、さらにサブルーチンを呼び出すためのマシン語を学びましょう。

解 説

ニーモニック： `CALL nn'`
(`nn'` は2バイト数値)

マシンコード： `CD n' n` *上下位逆転に注意。

機能： PCの値(戻り番地)をスタックへプッシュした後、 $PC \leftarrow nn'$ を行なう。すなわち、`nn'` 番地からのサブルーチンを呼び出す。

たとえば次のような状況であるといたしましょう。



CPUが、`CD 00 E0` という3バイト命令を取り込むと、プログラムカウンタPCは次に実行すべきアドレスとして `D003H` 番地を示します。ここで、CPUは `CALL 0E000H` を実行します。すなわち、PCの現在値である2バイトデータ `D003H` をスタックにプッシュします(それに伴いSPは2減じられます)。そして初めて、PCに `E000H` をロードします。こうして、`E000H` 番地(ここからサブルーチンが始まる)へのジャンプが行なわれます。

さて、サブルーチンの最後は `RET` で終わらなくてはなりません。CPUが `RET` を実行すると、SPの示すアドレスから2バイト分(戻り番地)をPCへポップし(それに伴ってSPの値は2増える)、かくして、`CALL 0E000H` の次の番地 `D003H` に復帰できるのです。

以上がサブルーチンの実行原理なのです。スタックに戻り番地が格納されている点が重要です。ですから、私たちがサブルーチン中 `LD SP, nn'` などを実行して勝手にスタックの位置を変更してしまったら、どうなるかわかりですね。戻り番地のデータが失われて、

RET を実行しても戻れなくなってしまう。最悪な場合は、「暴走」となる訳ですね。このように、ユーザープログラム内でのスタック操作にはくれぐれも注意が必要です（本書では、意図的にスタック操作を避ける方針をとっています）。

サブルーチン実行の実験をいたします。まず、CLEAR &HD000 で、マシン語フリーエリアを確保して下さい。次に、D000H番地から次のプログラムを入力して下さい。

《サブルーチン・コールの実験》

| アドレス | マシンコード | ニーモニック | コ メ ン ト |
|------|--------|------------|------------|
| D000 | CD0212 | CALL 1202H | サブルーチンをコール |
| D003 | C9 | RET | プログラムの停止 |

わずか4バイトのプログラムですからすぐ入力できますね。できましたら、*G D000 で実行して下さい。

```

MON
*G D000
D000
*

```

結果は上のようなはずですが、いかがですか？ 実は、1202H番地から始まるサブルーチンは、HLレジスタペアの内容を16進4桁で表示するものです。私たちは前節で、*G D000 を行なうと、HLレジスタペアは D000H に設定されることを学びましたね。この実験はそのことを証明しています。

HLレジスタペアの内容を16進4桁で画面表示させる処理は、かなり複雑そうですね。このように役立つサブルーチンが、モニターやBASICインタプリターのシステムプログラム中にたくさん用意されています。これをシステムサブルーチンとよびます。システムサブルーチンを上手に利用すると、複雑な処理を少ないバイト数で実現できます（上がその例です！）。しかし、そのためにはモニターやBASICインタプリターを解析して知識を得る必要があります。これは本書のページ数で説明することは不可能です。興味のある読者は自らの

手で、システムプログラムの解析に挑戦して下さい。そこはZ80のマシン語（とくにX1のマシン語）を勉強するための生きた教材そのものです。

3-23/第3章を終えるにあたって

3-17節以降、私たちは「スタック」というものの意味と働きについて理解を深めてきました。現在の目でもう1度「画面反転プログラム」を見直してみましょう。

図3-28 《画面反転サブルーチン》

| アドレス | マシンコード | ニーモニック |
|------|--------|--------------|
| D000 | 010020 | LD BC, 2000H |
| D003 | ED78 | IN A, (C) |
| D005 | CBDF | SET 3, A |
| D007 | ED79 | OUT (C), A |
| D009 | 03 | INC BC |
| D00A | 78 | LD A, B |
| D00B | FE23 | CP 23H |
| D00D | 38F4 | JR C, 0D003H |
| D00F | 79 | LD A, C |
| D010 | FEE8 | CP 0E8H |
| D012 | 38EF | JR C, 0D003H |
| D014 | C9 | RET |

いかがですか？ 今の私たちは、これがマシン語サブルーチンであることを明確に理解できますね。しかも、それは相対アドレスによりリロケートブルになっていますから、任意の本体プログラムの中に組み込み、呼び出すことができますね。これらのことがわかるようになったことは、マシン語に対する私たちの実力向上を示しています！ 自信を持って下さい！

次章からは、いよいよマシン語による高速ゲームの作成を始めます。画面反転サブルーチンも応用されます。楽しみにしていて下さい。

■ 第4章 マシン語ゲームに挑戦！ ■



■第4章 マシン語ゲームに挑戦！

4-1 / ゲームの選定

第3章で、私たちはテレビ画面をマシン語で制御する方法をマスターしました。本章では、いよいよマシン語ゲームに挑戦です。

どんなゲームを採用するか考えたのですが、ブロック崩し型のゲームや、インベーダー型のゲームは、よく例として採り上げられているようですから、本書では、「追いかっこ型」のゲームを材料にいたします。「追いかっこ型」ゲームの代表は、「パックマン」や「平安京エイリアン」ですが、ここでは、映画『トロン』に題材をとり、「トロン・ゲーム」を考えてみましょう。

余談になりますが、『トロン』を以前見た時、「面白い題だな」と思っていました。それが、X1でBASICを勉強してから、「TRON」というコマンドに出会い、このコマンドを題名にしたのがわかりました。ついでながら、TRON というのは、Trace On の略ですね。これを実行すると、BASICインタプリターが今、行番号いくつの所を実行しているか追跡することができます。TRONモードからの脱出は、TROFF を実行すればよいのです。

さて、映画『トロン』では、レーザー分解装置により、コンピューター世界に入ってしまった主人公が、バイクに乗って追跡ゲームをさせられるシーンがありました（「ライトサイクルゲーム」というのだそうです）。この部分をX1でゲーム化してみましょう。

まず、ゲームの構成を把むために、オールBASICで作成してみました。

4-2 / オールBASIC版作成にあたって

オールBASIC版の目的ですが、これはあくまで全体をつかむためのテスト用として作成します。従って、プログラムの構造を明確にすることが第1目的となります。

高速化は後に、マシン語版で考えることにし、速度を犠牲に少しでも、プログラムリストを見やすくすることに努めました。マルチステートメントはなるべく避け、ラベルと注釈（REM ないしは、その省略形の）を多用しました。これらは、速度を遅くする大きな原因ですが、敢えてこういたします。

次にプログラムリストを掲げます。

図4-1 BASICリスト

```

10 /
20 /
30 / TRON GAME ver.1
40 /
50 / by Yasuhiro Shimizu
60 /
70 / 1983.10.15
80 /
90 /
100 /
110 / ショキカ
120 /
130 INIT :WIDTH 40 :CLS 4 :CLICK OFF :TEMPO 200
140 DEFINT A-Z
150 /
160 GOSUB "オープニング"
170 GOSUB "キャラクタータイク"
180 /
190 / スウチ / ショキセツタイ
200 /
210 DX=-1 :DY=0
220 LI=ASC("-") :B=120
230 /
240 DEF FNT(X,Y)=PEEK@(&H3000+X+Y*40)
250 DEF FNS(U,W)=PEEK@(&H3000+U+W*40)
260 /
270 / セツマイ
280 /
290 CLS
300 LOCATE 10,0 :COLOR 4 :CSIZE 3 :PRINT#0 "TRON GAME" :CSIZE 0
310 LOCATE 13,4 :COLOR 6 :PRINT "TRON:"
320 LOCATE 18,4 :COLOR 7 :CGEN 1 :PRINT CHR$(120) :CGEN 0
330 LOCATE 13,6 :COLOR 2 :PRINT "SARK:"
340 LOCATE 18,6 :COLOR 7 :CGEN 1 :PRINT CHR$(220) :CGEN 0
350 LOCATE 16,8 :PRINT "KEY"
360 LOCATE 17,10 :PRINT "8"
370 LOCATE 16,11 :PRINT "4 6"
380 LOCATE 17,12 :PRINT "2"
390 LOCATE 13,14 :COLOR 3 :PRINT "10 POINT MATCH"
400 LOCATE 13,18 :COLOR 7 :PRINT "HIT"
410 LOCATE 17,18 :CFLASH 1 :PRINT "RETURN KEY" :CFLASH 0
420 REPEAT
430 I$=INKEY$
440 UNTIL I$=CHR$(13)
450 /
460 / ケーダ スタート
470 /
480 T=0 :S=0 :G=0
490 CLS
500 LOCATE 8,8 :COLOR 5 :CSIZE 3
510 PRINT#0 "GAME START" :CSIZE 0
520 MUSIC "04R2C3DEFGAB+C"
530 /
1000 / カメン ツクリ
1010 /
1020 CONSOLE 1,24 :CLS :CONSOLE
1030 COLOR 4
1040 LINE (0,1)-(38,1), "-"
1050 LINE (1,23)-(38,23), "-"
1060 LINE (0,2)-(0,22), "|"
1070 LINE (39,2)-(39,22), "|"
1080 LOCATE 0,1 :PRINT "r"
1090 LOCATE 39,1 :PRINT "l"
1100 LOCATE 0,23 :PRINT "L"
1110 LOCATE 39,23 :PRINT "J";
1120 LOCATE 1,0 :COLOR 6 :PRINT "TRON:";
1130 LOCATE 20,0 :COLOR 2 :PRINT "SARK:";
1140 /
1150 / シュウハツ イチ / ケツタイ
1160 /
1170 RANDOMIZE TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210 U=INT(RND*36)+2
1220 V=INT(RND*19)+3
1230 UNTIL (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DU=INT(RND*2)*2-1 :DV=0
1250 CGEN 1 :COLOR 7
1260 LOCATE X,Y :PRINT#0 CHR$(120)

```

```

1270 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1280 FOR I=1 TO 3 :BEEP :NEXT
1290 /
2000 /   メイン・ループ   /
2010 /
2020 M=DU :N=DV :F=1
2030 I=INT(RND*20)+1 :IF I<=4 THEN 2100
2040 /
2050 IF Y<V AND FNS(U,V-1)=ASC(" ") THEN DU= 0 :DV=-1 :GOTO 2150
2060 IF Y>V AND FNS(U,V+1)=ASC(" ") THEN DU= 0 :DV= 1 :GOTO 2150
2070 IF X<U AND FNS(U-1,V)=ASC(" ") THEN DU=-1 :DV= 0 :GOTO 2150
2080 IF X>U AND FNS(U+1,V)=ASC(" ") THEN DU= 1 :DV= 0 :GOTO 2150
2090 /
2100 IF FNS(U,V-1)=ASC(" ") THEN DU= 0 :DV=-1 :GOTO 2150
2110 IF FNS(U,V+1)=ASC(" ") THEN DU= 0 :DV= 1 :GOTO 2150
2120 IF FNS(U-1,V)=ASC(" ") THEN DU=-1 :DV= 0 :GOTO 2150
2130 IF FNS(U+1,V)=ASC(" ") THEN DU= 1 :DV= 0
2140 /
2150 IF DV=-1 THEN GOSUB 4000 :GOTO 2190
2160 IF DV=1 THEN GOSUB 4100 :GOTO 2190
2170 IF DU=-1 THEN GOSUB 4200 :GOTO 2190
2180 GOSUB 4300
2190 LOCATE U,V :COLOR 2 :CGEN 1
2200 PRINT CHR$(LI);
2210 U=U+DU :V=V+DV
2220 IF FNS(U,V)=ASC(" ") THEN 2280
2230 /
2240 LOCATE U,V :COLOR 2 :CGEN 0
2250 PRINT "X";
2260 T=T+1 :MUSIC "04R3C1DEFGAB+CR3" :GOSUB "カチ ノ ハンタイ" :IF H$<>" " THEN 3000 ELSE GOTO 1000
2270 /
2280 LOCATE U,V :COLOR 7
2290 PRINT CHR$(B) :CGEN 0
2300 /
2310 M=DX :N=DY :F=0
2320 I=STICK(0)
2330 IF I=8 THEN DX= 0 :DY=-1 :GOTO 2370
2340 IF I=2 THEN DX= 0 :DY= 1 :GOTO 2370
2350 IF I=4 THEN DX=-1 :DY= 0 :GOTO 2370
2360 IF I=6 THEN DX= 1 :DY= 0
2370 IF M*DX<0 OR N*DY<0 THEN DX=-DX :DY=-DY
2380 /
2390 IF DY=-1 THEN GOSUB 4000 :GOTO 2430
2400 IF DY= 1 THEN GOSUB 4100 :GOTO 2430
2410 IF DX=-1 THEN GOSUB 4200 :GOTO 2430
2420 GOSUB 4300
2430 LOCATE X,Y :COLOR 6 :CGEN 1
2440 PRINT CHR$(LI);
2450 X=X+DX :Y=Y+DY
2460 IF FNT(X,Y)=ASC(" ") THEN 2520
2470 /
2480 LOCATE X,Y :COLOR 6 :CGEN 0
2490 PRINT "X";
2500 S=S+1 :MUSIC "04R3+C1BAGFEDCR3" :GOSUB "カチ ノ ハンタイ" :IF H$<>" " THEN 3000 ELSE GOTO 1000
2510 /
2520 LOCATE X,Y :COLOR 7
2530 PRINT CHR$(B) :CGEN 0
2540 /
2550 GOTO 2000
2560 /
2570 /
3000 /   ゲームオーバー   /
3010 /
3020 FOR I=0 TO 1000
3030 A=PEEK$( &H2000+I)
3040 A=(A OR &B1000)
3050 POKE$ &H2000+I,A
3060 NEXT
3070 /
3080 LOCATE 8,8 :COLOR 7 :CSIZE 2
3090 PRINT#0 " GAME!" +H$+" " :CSIZE 0
3100 IF H$="TRON" THEN MUSIC "04C2DEFGAB+C"
3110 IF H$="SARK" THEN MUSIC "04+C2BAGFEDC"
3120 /
3130 LOCATE 5,12 :PRINT "DO YOU WANT REPLAY [ Y or N ]"
3140 REPEAT
3150 I$=INKEY$
3160 UNTIL I$="Y" OR I$="N"
3170 /
3180 IF I$="Y" THEN 460   ;'   ——> ゲーム スタート
3190 INIT :CLS
3200 LOCATE 15,10 :PRINT "オツカレグマ ♥"
3210 END
3220 /
3980 /   サンプル・シーン   /
3990 /
4000 /   ウェニイフ   /
4010 /

```

```

4020 LI=ASC("I")
4030 IF M=-1 THEN LI=ASC("L")
4040 IF M= 1 THEN LI=ASC("J")
4050 IF F=0 THEN B=100 ELSE B=200
4060 RETURN
4070 /
4100 /      シタ ニ イク      /
4110 /
4120 LI=ASC("I")
4130 IF M=-1 THEN LI=ASC("L")
4140 IF M= 1 THEN LI=ASC("J")
4150 IF F=0 THEN B=110 ELSE B=210
4160 RETURN
4170 /
4200 /      ヒタリ ニ イク      /
4210 /
4220 LI=ASC("-")
4230 IF N=-1 THEN LI=ASC("J")
4240 IF N= 1 THEN LI=ASC("J")
4250 IF F=0 THEN B=120 ELSE B=220
4260 RETURN
4270 /
4300 /      ミキ ニ イク      /
4310 /
4320 LI=ASC("-")
4330 IF N=-1 THEN LI=ASC("L")
4340 IF N= 1 THEN LI=ASC("L")
4350 IF F=0 THEN B=130 ELSE B=230
4360 RETURN
4370 /
4380 /
4980 /
4990 /
5000 LABEL "カチ ノ ハンタイ"
5010 /
5020 /
5030 /
5040 GOSUB "スコア"
5050 IF G=1 THEN 5140
5060 /
5070 H$=""
5080 IF T=10 THEN H$="TRON"
5090 IF S=10 THEN H$="SARK"
5100 RETURN
5110 /
5120 /      シュース ノ ショリ      /
5130 /
5140 H$=""
5150 IF T=S+2 THEN H$="TRON"
5160 IF S=T+2 THEN H$="SARK"
5170 RETURN
5180 /
5190 /
5980 /
5990 /
6000 LABEL "スコア"
6010 /
6020 /
6030 /
6040 MUSIC "05G1+C"
6050 LOCATE 8,0 :COLOR 6
6060 PRINTUSING "###";T;
6070 LOCATE 27,0 :COLOR 2
6080 PRINTUSING "###";S;
6090 /
6100 LOCATE 12,0 :PRINT " ";
6110 IF T>=9 AND S>=9 AND T=S THEN G=1:LOCATE 12,0 :COLOR 7 :CFLASH 1 :PRINT "シュース"; :CFLASH 0
:FOR I=1 TO 2 :BEEP 1 :PAUSE 5 :BEEP 0 :PAUSE 5 :NEXT
6120 /
6130 RETURN
6140 /
6980 /
6990 /
7000 LABEL "オープニング"
7010 /
7020 /
7030 /
7040 FOR I=1 TO 184
7050 COLOR (I MOD 6)+1 :PRINT "TRON ";
7060 NEXT
7070 PAUSE 20
7080 RETURN
7090 /
7100 /
7980 /
7990 /
8000 LABEL "キャラクター・タイキ"
8010 /

```

```

8020 / ██████████
8030 /
8040 DEFCHR$(100)=HEXCHR$("0018183C3C181818667E7E3C3C7E7E66666666000066667E")
8050 DEFCHR$(110)=HEXCHR$("1818183C3C181800667E7E3C3C7E7E667E6666660000666666")
8060 DEFCHR$(120)=HEXCHR$("000E7F1818000000000E7FFFFF7000000000E7E7E70000")
8070 DEFCHR$(130)=HEXCHR$("0070FE181800000000070FEFFFF7000000000E7E7E70000")
8080 /
8090 DEFCHR$(200)=HEXCHR$("0018183C3C181818666666000066667E0000000000000018")
8100 DEFCHR$(210)=HEXCHR$("1818183C3C1818007E666600006666661800000000000000")
8110 DEFCHR$(220)=HEXCHR$("000E7F181800000000000E7E7E7000000000000000000")
8120 DEFCHR$(230)=HEXCHR$("0070FE181800000000000E7E7E7000000000000000000")
8130 /
8140 DEFCHR$(&H90)=HEXCHR$("000000FFFF000000000000FFFF000000000000FFFF000000")
8150 DEFCHR$(&H91)=HEXCHR$("1818181818181818181818181818181818181818181818")
8160 DEFCHR$(&H97)=HEXCHR$("000000F8F8181818000000F8F8181818000000F8F8181818")
8170 DEFCHR$(&H98)=HEXCHR$("181818F8F8000000181818F8F8000000181818F8F8000000")
8180 DEFCHR$(&H99)=HEXCHR$("1818181F1F0000001818181F1F0000001818181F1F000000")
8190 DEFCHR$(&H9A)=HEXCHR$("0000001F1F1818180000001F1F1818180000001F1F181818")
8200 /
8210 RETURN
8220 /
9000 / XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9010 / * * * * *
9020 / * Sample Game for X1 マシンに ニュモン *
9030 / * * * * *
9040 / * all BASIC version *
9050 / * * * * *
9060 / XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```


4-3 / ゲーム作成の注意点

私としては、できる限り見やすいリストにしたつもりなので、詳細はリストを見ていただければわかると思いますが、いくつか注意すべき点がありますので以下に挙げます。

- ① ゲームをRUNする前に、コンピューターの状態はどうなっているか不明なので、最初に必ず「初期化」を行ないましょう。X1のBASICには、画面初期化のための便利なコマンド `INIT` がありますので、これを使用しました(130行)。
- ② 140行は、すべての変数を整数型に宣言しています。ゲームでは、グラフィック画面でコンピューター・グラフィックスを行なう時以外は、ほとんど実数型変数は不要ですから、このようにしておくと、速度が上がります。
- ③ 160行の「オープニング」について注意しておきます。よく経験するのは、RUNした後、画面に何の変化も起きない時間が数秒間続くケースです。これでは、プログラムがちゃんと動き始めたのか、何か原因不明のトラブルが生じたのかわかりませんね。特に、今の場合、170行で「ユーザーキャラクターの定義」をしていて、これは一般に、結構時間がかかるので、「オープニング」で素早く反応しておくことは大切だと思います。ただし、今の場合はユーザーキャラクターの数も少ないので、「オープニング」には「待ち」を入れておきます。
- ④ 240行と250行では、`FNT(X, Y)`、`FNS(V, W)` というユーザー用の関数を定義しています。トロン(操作する私たち)とサーク(コンピューター側)の移動を決定するためには、画面上の進みたい位置に障害物があるか否かを頻繁に判定しなくてはなりません。そこで、そのためのVRAMのASCIIコード読みとり関数を定義した訳です。
- ⑤ 300行に、`PRINT#0` とありますが、このコマンドは、倍文字やユーザー定義文字を表示するためのものです。
- ⑥ 420行~440行は、リターンキー(ASCIIコード13)のキー待ちです。
- ⑦ 480行は、ゲーム内での1サイクル開始のための数値初期化です。Tはトロンのスコア、Sはサークのスコアです。Gはジュース判定用の変数(フラグの働き!)で、0ならジュースでなく、1ならジュースを意味します。
- ⑧ 1020行は、最上行にあるスコアを残して画面を消去するテクニックです。

- ⑨ 1170行は、乱数系列を内蔵タイマーを利用して変える方法です。よくゲームの本に、
RANDOMIZE TIME としてあるのを見ますが、X1では注意が必要です。マ
ニュアルを見るとわかりますが、RANDOMIZE の引数は、-32768~65
535 まで、内蔵タイマー変数 TIME の値は、0~86399 と変化します
ね。ですから、65535を越えた時の処理が必要です。TIME-20864 として
おくと、この値は、-20864~65535 の範囲で変化して、RANDOMIZE
の引数範囲に納まり、Overflow エラーを生じません。
- ⑩ 1200~1230行は、サークの初期位置をトロンの初期位置から、距離5以上離すた
めの処理です（ピタゴラスの定理を思い出して下さい！）。
- ⑪ 1280行は、出発合図のベルです。
- ⑫ いよいよ、メイン・ループへ突入です。まず、2020行は、トロンのサーク共用のキャ
ラクター選定ルーチン（4000~4360行）へのデータ引き渡しの準備です。サブルー
チンでは共用の変数として、MとNを使用しています。Fは、トロンのサークの識別コード
で0がトロンの、1がサークです（トロンのサークフラグ！）。
- ⑬ 2030行では、サークの行動に偶然性を持たせるための乱数設定をします。
- ⑭ 2050~2080行は、サークがトロンの近づくための簡単なアルゴリズムです。
- ⑮ 2100~2130行は、サークがトロンと無関係に移動方向を決定するための部分で
す。トロンのいる方向に障害物がある時、および乱数が一定の値を示す時に、このルーチン
が実行されます。
- ⑯ 2150~2180行は、サークの軌跡およびバイクのキャラクターを選定しています。
- ⑰ 2190~2290行は、サークのキャラクターを表示する部分です。障害物への衝突を
判定し（2220行）、この時には、トロンのスコアを+（プラス）して、音楽をならし、
「勝負判定ルーチン」（5000行）へ飛びます。
- ⑱ 2310行は、2020行と同様に、サブルーチンへのデータ引き渡しです。トロンの
サークフラグFを0にして、トロンのモードにします。
- ⑲ 2320~2360行は、キースキャンです。テンキーのどれが押されたかを判定します。
- ⑳ 2370行は地味ですが重要な処理で、誤動作対策です。この処理の効果は、この行を消
してゲームしてみるとすぐにわかります。
- ㉑ 2390~2420行は、トロンの軌跡およびバイクのキャラクター選定です。
- ㉒ 2430~2530行は、トロンのキャラクター表示を行いません。衝突の時は、サーク

のスコアを+（プラス）し、音楽をならして、「勝負判定ルーチン」へ飛びます。

㊸ 2550行で、ループさせています。

㊹ 3000～3210行は、ゲーム終了の処理で、「勝負判定ルーチン」から、ゲーム終了と判断されると、ここへ来ます。（終了判定は、H\$という文字変数が空でないことでなされる。）特に、3020～3060行は、1画面反転処理です。この部分をマシン語化することはもうできますね！

㊺ 4000～8210行には、サブルーチンをまとめておきました。少しでも速度を上げるために、移動方向によるキャラクター選定ルーチン（4000～4360行）にはLABEL文を使いませんでした。

本当は効果音など、もっと音楽を工夫すると一層面白くなるのですが、手もとに音楽のデータ（楽譜等）がないため、ドレミファソラシドでガマンしています。読者の皆様、是非工夫してみてください。また、サークがトロンを追跡するアルゴリズムも、もっと凝ったものを考えてみてください。

以上で、オールBASIC版の説明を終えますが、これは、後のマシン語版の土台となりますので、是非入力して、遊んでみることをお勧めします。マシン語で少し大きめのプログラムを組むには、プログラムの構造がしっかり理解できていなくてはなりません。BASICプログラムのように「いきあたりばったり」的な作り方をすると大変に非能率的です。このオールBASIC版を材料にして、プログラムの構造を頭に入れておいて下さい。また、オールBASIC版は速度が遅いのがおわかりになると思いますが、プログラム中どこが速度を遅くしている原因であるかを考えてみてください。その部分こそ「マシン語化」のターゲットなのですから！

4-4 / マシン語化にあたって

オールBASIC版の「トロン・ゲーム」いかがでしたか？ トロンとサークの動きが、いかにもモタモタしていますね。本節からは、マシン語による高速化を考えてゆきましょう！

マシン語ゲームを作成するにあたり、基本的なことをまず考えておきます。オールマシン語版ができれば、もち論文句なしですが、後にマシン語プログラムを作成する段階になるとわかるように、BASICに比べて、マシン語によるプログラム作成の苦労は倍化いたします。また、BASICインタプリターが行なってくれていた「システムの初期化」—— 本格的な

周辺機器の制御 —— を自前でやらねばなりません。このようなことは、本書のページ数では納まらないことは明らかで、またX1のハードウェアの核心部分の理解も不可欠になります。そこで、オールマシン語（IPL起動型とよく言う）のプログラムは断念いたします。

ではどうするかというと、中庸をとって、画面作りや、キャラクター定義など、ゲームの核心から離れた飾り的部分は、BASICで作成し、高速を要求されるメイン・ループの部分をマシン語サブルーチンにすることを考えます。このような作り方を、BASICとマシン語のリンクとよびます。

4-5/BASICとマシン語のリンクの実際

マシン語サブルーチンを、BASIC（のメインルーチン）とリンクさせる時には、マシン語部分の実行は、「BASICからマシン語サブルーチンを呼び出す」形で行なわれます。前章までのように、モニターを起動して、*G アドレス でないことは、もうおわかりですね。

マシン語サブルーチンの呼び出しは、当然 GOSUB では行なえません。GOSUBは行番号付きのBASICサブルーチンの呼び出しのための命令でした。では、どうすればよいか？ 困った時はマニュアルを熟読いたしましょう！

マニュアル178ページに、次のように書いてあります。

BASICより機械語サブルーチンを呼び出す命令には、
CALL命令とUSR命令があります。

私たちは、この2つのBASIC命令についてよく理解しておかねばなりません（次節から3節にわたり詳しく見ます）。

ここでは、もう少し基本的なことを考えておきましょう。BASIC部分と、マシン語部分が混在するのですから、「どこまでがBASICで、どこからがマシン語」かを明確にしておく必要があります。すなわち、私たちは前章で確認したように、「マシン語フリーエリアの確保」を行なっておく必要があります。このためには、CLEAR アドレス というBASIC命令を用いるのでしたね。こうして、指定したアドレスから、FEFFH番地までが、私たちのマシン語サブルーチンを格納しておく場所になるのです。

次に、そもそもマシン語サブルーチンはどのようにして書き込むのか？ について考えておきます。2つの方法があると思います。

第1の方法は、あらかじめモニターを起動して、前章までのようなやり方で、マシン語部分をメモリーに書き込んでおくというものです。

第2の方法は、BASICプログラムの中に、DATA文でマシン語部分を書いておき、プログラム中、READ と POKE により、メモリーに書き込む方法です。

本書では、マシン語部分の作成中は、テストも込めて第1の方法をとり、完成した後、第2の方法で1本のプログラムにまとめる方針でいきます。

以上の諸点よろしいですか？ ではお約束したように、「BASICからマシン語サブルーチン呼び出す方法」から見ていきましょう。

4-6/BASICのCALL命令

BASICの CALL 命令は、CALL マシン語サブルーチンの先頭アドレス という書式で用います。マシン語のニーモニックと同じ名称ですが、これはあくまでBASICのコマンドとしてのCALLですよ！

早速実験してみましょう。あらかじめ、CLEAR &HD000 により、マシン語フリーエリア(D000H~FEFFH番地)を確保しておいて下さい。マシン語サブルーチンの材料としては、前章で作成した「画面反転サブルーチン」を使いましょう。

MON でモニターを起動し、マシン語をメモリーに書き込みます。フリーエリア内ならどこでも構いませんが、3-23節に掲げたリストはD000H番地からにしてありますから、まあこの番地からにしましょう。D000H~D014H番地に書き込み完了しましたか？

*R コマンドで、BASICの管理下に戻ります。では、いよいよ実行です！

CALL &HD000

を実行して下さい。予定通りの結果が出ましたか？ 入力ミスがなければ、見事に白黒反転が起こるはずです。

こうして私たちは、BASIC からマシン語サブルーチンを実行する方法を1つ身につけたことになります。

4-7 /USR関数の理解をめざしてー1ー

次に2つ目の USR関数 の説明をいたします。まず、マニュアル178ページを御覧ください。かなり詳しい説明がありますね。ですから、マシン語について知識のある方でしたら、マニュアルの説明だけできっとおわかりになるでしょう。

しかし、マシン語を勉強し始めた当時の私には、この説明は難かしすぎました。実験しようにも、どうしたらよいかわかりませんでした。この体験、おそらく本書の読者であるマシン語初心者の方とも共有できると思います。そこで、本書では、2つの実験を用意いたしましたので、実験を通して理解を深めていただきたいと思います。

まず、CALL命令と本質的に違う2点を確認しておきます。

確認1 : USRは、関数である。 CALLは、関数ではない。

確認2 : CALLはいきなりで使えるが、USRは実行に先立って
DEFUSR で定義しておかねばならない。

関数については、おわかりですか？ 代表的なものには、SINがありますね。

これは、 SIN (数値又は数式) という形で用います。すると関数ですから、与えた数値又は数式のサイン(正弦)を計算して出力しますね。たとえば、 $Y = \sin(\pi/6)$ とすると、変数Yに $\pi/6$ ラジアン(=0.5)のサイン(=0.5)が代入されます。

確認1は、USRもこのような「関数」の一種だと言っているのです。従って、

$Y = \text{USR}(\text{データ})$

のように用いると、データが加工されて、変数Yに代入されます。

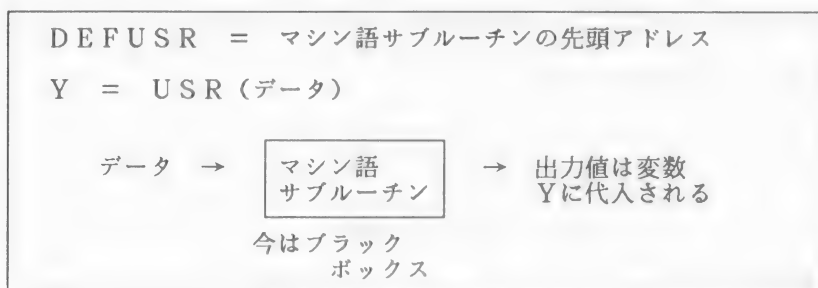
今、「データが加工されて」と述べました。では、どのようにして加工されるのでしょうか。SIN関数の場合は、私たちは、入力した数値を加工する三角関数の法則を知っている訳です。では、USR関数の場合は、どのような法則なのでしょう。

これを理解する大枠が、確認2の内容なのです。すなわち、大ざっぱに言うと、

DEFUSR であらかじめ指定しておいたアドレスから始まる
マシン語サブルーチンによって加工される。

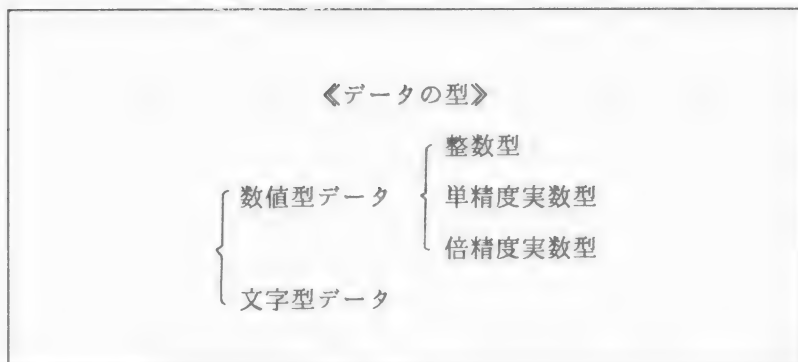
というのが答えです。

内部の詳細は後まわしにして、大枠をつかむと次のような図式になります。



まず大ざっぱに理解していただけましたか？ では次に、データを引き渡す仕組みを見てまいりましょう。

入力するデータについて考察します。BASICでは、データの型を4種類に分けています。



これらのデータ（のいずれか）を、USR関数に入力いたしますと、まず、Aレジスタにデータの型の識別コードが格納されます。

《Aレジスタに入る値》

| | |
|--------------|-------|
| 整数型データのとき | 0 2 H |
| 単精度実数型データのとき | 0 5 H |
| 倍精度実数型データのとき | 0 8 H |
| 文字型データのとき | 0 3 H |

まず、このことを実験で確認してゆきましょう。Aレジスタの値は直接見ることはできませんから、メモリーに移して、PEEK関数で読むことにいたします。すなわち、マシン語サブルーチンの内容としては、Aレジスタの内容をメモリー（ここでは仮にE000H番地としておく）に移す作業を担当してもらいましょう。例によって、CLEAR &HD000 でフリーエリアを確保し、仮に、D000H番地からマシン語サブルーチンを書き込みます。

《USR関数のテスト1》

| アドレス | マシンコード | ニーモニック |
|------|--------|----------------|
| D000 | 3200E0 | LD (0E000H), A |
| D003 | C9 | RET |

次に、テスト用のメインルーチンをBASICで組みます。

図4-2 テストプログラム

```

10 REM —— USR カンスト A レジスタ ——
20 /
30 DEFUSR=&HD000
40 /
50 CLS
60 LOCATE 0,3
70 PRINT "ツキノトブレカヲシタイシテ、クダサイ。"
80 PRINT
90 PRINT
100 PRINT "セイスウ カクタ _____> 1
110 PRINT "タンセイト シツスウカクタ _____> 2
120 PRINT "ハクイセイト シツスウカクタ _____> 3
130 PRINT "モシカクタ _____> 4
140 LOCATE 0,0 :BEEP :INPUT "USR カンスト ニ アタイル テータ ノ シュルイ ハ ";N
150 LOCATE 0,13
160 ON N GOTO 170,180,190,230
170 BEEP :INPUT "テータ ハ ";A% :B=USR(A%) :GOTO 200
180 BEEP :INPUT "テータ ハ ";A! :B=USR(A!) :GOTO 200
190 BEEP :INPUT "テータ ハ ";A# :B=USR(A#)
200 AREG=PEEK(&HE000)
210 BEEP :PRINT "A レジスタ = ";RIGHT$("0"+HEX$(AREG),2)+"H", "USR シュツリョク = ";B
220 END
230 BEEP :INPUT "テータ ハ ";A$ :B$=USR(A$)
240 AREG=PEEK(&HE000)
250 BEEP :PRINT "A レジスタ = ";RIGHT$("0"+HEX$(AREG),2)+"H", "USR シュツリョク = ";B$
260 END

```

このBASICプログラムを用いて、実験して下さい。実験例を下に掲げます。

| | |
|------------|---------------------------|
| [例] 整数型データ | 10 ⇒ Aレジスタ = 02H |
| | USR出力 = 10 |
| 単精度実数型データ | .5 ⇒ Aレジスタ = 05H |
| | USR出力 = .5 |
| 倍精度実数型データ | .0000000005 ⇒ Aレジスタ = 08H |
| | USR出力 = 5E-10 |
| 文字型データ | "X" ⇒ Aレジスタ = 03H |
| | USR出力 = X (文字型のX) |

以上で第1の実験を終えます。USR関数とAレジスタの関係おわかりになりましたか？

4-8 / USR関数の理解をめざして-2-

次にいよいよUSR関数の核心部分に迫ります。USR関数へ渡すデータは、どのように格納されるのか？ という点です。

これもデータの4つの型ですべて異なりますが、本書では、整数型データの場合のみを扱います。他の3つの型についてはマニュアルの説明と以下に掲げるプログラムを参考にして、皆さん自身で実験法を考えて下さい。

整数型データの場合は、HLレジスタペアが、格納アドレスを示します。具体的には、整数値を2バイトのデータと見なして、

| | |
|-------|-----------------|
| 下位バイト | ← HLで示されるアドレス |
| 上位バイト | ← HLで示されるアドレス+1 |

と格納されます（上下位逆転していることに注意！）。では実験にかかりましょう。今度のマシン語サブルーチンは、HLレジスタペアの値、及び、HLレジスタペアの示すアドレスの内容、HL+1の示すアドレスの内容をメモリーのE000H番地から保存しておくものです（CLEAR &HD000 を忘れずに！）。

《USR関数のテスト2》

| アドレス | マシンコード | ニーモニック |
|------|--------|----------------|
| D000 | 2200E0 | LD (0E000H),HL |
| D003 | 7E | LD A,(HL) |
| D004 | 3202E0 | LD (0E002H),A |
| D007 | 23 | INC HL |
| D008 | 7E | LD A,(HL) |
| D009 | 3203E0 | LD (0E003H),A |
| D00C | C9 | RET |

このサブルーチンを実行すると、

メモリーのE000H番地 ← Lレジスタの値

E 0 0 1 H 番地 ← Hレジスタの値
 E 0 0 2 H 番地 ← HLレジスタの示すメモリーの値
 E 0 0 3 H 番地 ← HLレジスタ+1の示すメモリーの値

とデータが移動することはおわかりですね。（上下位逆転が起こっていますから注意！）
 さて、テスト用のBASICプログラムを掲げます。

図4-3

```

10 REM —— USR カンフ テノ セイスウ データ ノ カクノ ——
20 /
30 DEFUSR=&HD000
40 /
50 WIDTH 40 :CLS
60 BEEP :INPUT "セイスウ データ = ";A%
70 B=USR(A%)
80 HREG=PEEK(&HE001) :H%=RIGHT$("0"+HEX$(HREG),2)
90 LREG=PEEK(&HE000) :L%=RIGHT$("0"+HEX$(LREG),2)
100 HLMEM=PEEK(&HE002) :ML%=RIGHT$("0"+HEX$(HLMEM),2)
110 HLNXT=PEEK(&HE003) :MH%=RIGHT$("0"+HEX$(HLNXT),2)
120 PRINT :BEEP
130 PRINT "HL レジスタ = ";H%;L%;"H"
140 PRINT
150 PRINT "HL ノ シメス メモリー ノ ナイヨウ = ";ML%;"H"
160 PRINT "HL+1 ノ シメス メモリー ノ ナイヨウ = ";MH%;"H"
170 PRINT
180 PRINT "USR シュツリョク = ";B
190 END

```

このプログラムを走らせ、いろいろな整数値を入れて、結果を見て下さい。USR関数によるデータ格納の様子、御理解いただけましたか？

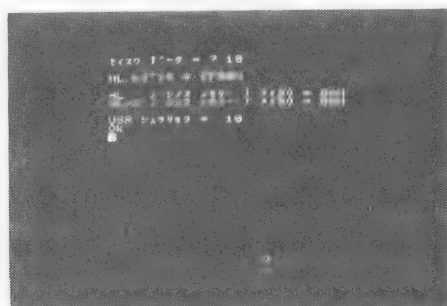


図4-4

以上、2つの実験で、USR関数により、マシン語サブルーチンへBASICメインルーチンからデータを引き渡す方法がわかりました。マシン語サブルーチンでは、このようにして、

データの型及びデータを知ることができます。

さて、実験プログラムではいずれも、受け取ったデータ自体をサブルーチン内で加工することとはしていません。まあ、理屈っぽく言うとな「何もしない」という加工をしていることになりますが、ともあれ、マシン語サブルーチンに入力されたデータをそのままメインルーチン側に返しています。USR関数の真髓を理解するには、本当はもう一步進んで「加工・出力のされ方」をもっと調べる必要があるでしょう。しかし、本章の目的は、あくまでマシン語学習の一端としてのゲームの作成であり、また、本書では、マシン語サブルーチンから（加工されて）返ってくるデータを、BASICメインルーチンで利用しない予定ですので、USR関数による出力値の考察は、これ以上しないことにいたします。興味のある読者は是非、実験を試みて下さい。

[注] 2番目の実験に関して一言注意しておきます。CLEAR &HD000 によりマシン語フリーエリアを確保してから実験しますと、HLレジスタペアの示すアドレス値は、おそらく CF00H となっているはずですが、実は今の場合、CF00H番地から CFFFH番地は、FAC (Floating-point Accumulator=浮動小数点アキュムレータ) というエリアとして使用されています。実数データを扱うには、多くのバイト数を要しますが、CPUの内部レジスタには限りがありますから、メモリー上の一部を実数計算用のアキュムレータと見なして使います。これがFACです。マニュアル183ページのメモリーマップを見るとわかるように、FACの開始アドレスは、(CLEAR文で指定したアドレス) - 100H と決められています。ですから、CLEAR &HD000 のとき、FACは CF00H番地から設定されることになりますね。

USR (数値型データ) により、マシン語サブルーチンへデータを引き渡す場合は、HLレジスタペアは、このFACの開始アドレスを示すことになります。従って私たちの実験でも、HLの値は CF00H となったのでした。

ところで、読者の中には MON によりモニターを起動して、CF00H番地と CF01H番地の2バイト分のメモリー内容を確認された方もいると思います。いかがでしたか？

多分、予期したデータ（入力した2バイト整数データ）はなかったはずですが、しかし、この現象は実験のミスではありません。理由は、マシン語サブルーチンを実行して、FACのデータを E000H番地以降へ退避させた後に、BASICインタプリターのシステムプ

プログラムがFACを書き換えてしまうからです。従って、FACのデータを保存しておくには、このようにサブルーチン内の処理で退避させておく必要があるのです。

システムがFACを書き換えると述べましたが、もちろん書き換え方にも「法則」があるはずです。しかし、この部分は現段階では秘密のベールに包まれていますね。読者自らの手で、いつかこれらの秘密のベールをはいでみて下さい。

4-9 / マシン語サブルーチンの配置

私たちは、2つのBASIC命令(CALLとUSR)の基本的使い方をマスターすることで、BASICメインルーチンからマシン語サブルーチンを実行することができるようになりました。

もはや、私たちの前には、大きな目標 —— トロンゲームのマシン語化 —— だけが残されるのみとなりました。取り組む準備はよろしいですか？ では、マシン語ゲーム作成の始まり、始まり……。

まず、トロンゲームのオールBASIC版のうち、どの部分をマシン語化するかを明確にしておきましょう。「メインループ」内を見て下さい。サークとトロンの行動決定・キャラクター選定のために、いかにもギコチナイ IF～THEN 文の条件判断が続いています。これは明らかに速度低下の原因です。従って、行番号 2020行～2250行、および2280行～2290行の部分を「サークの移動ルーチン」としてマシン語化します。同様に、2310行～2490行、および2520行～2530行を「トロンの移動ルーチン」としてマシン語に直します。これに伴い、4000行～4360行のサブルーチンはマシン語ルーチンに吸収します。2260行と2500行は、1サイクルの終了処理で「勝負判定」の部分ですから、ループの速度には大きく影響はないはずで、BASICのままにいたします。

また、「ゲームオーバー」ルーチン内の3020行～3060行は「画面反転」であり、BASICの FOR～NEXT ループでは、いかにもモタモタしていますから、前章で私たちが完成した「画面反転サブルーチン」で置き換えましょう。

こうして、「マシン語化」の目標がはっきりいたしました。

作成するマシン語サブルーチン

- (1) サークの移動ルーチン
(SARKMVと名づける)
- (2) トロンの移動ルーチン
(TRONMVと名づける)
- (3) 画面反転ルーチン
(CREVと名づける)


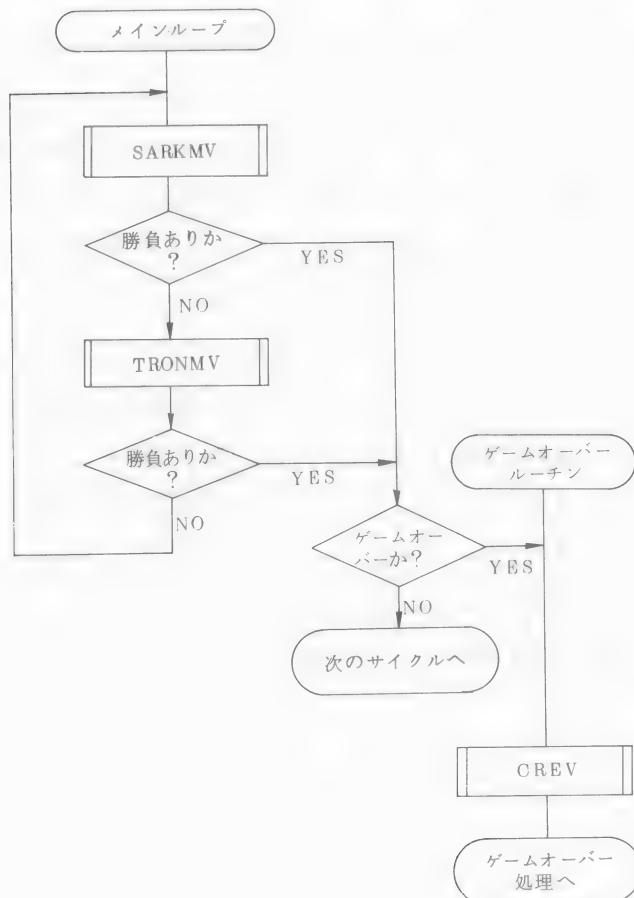
メインループのフローチャートは次のようになります。各マシン語サブルーチン ( で表示) が、どのように位置づけられているかを頭に入れて下さい。

図4-5



4-10/サブルーチンの仕様を決める！

次にする仕事は、3つのサブルーチンの仕様（どんな方法で、どんな順序ですか）の決定です。オールBASIC版を参考に、各ルーチンでどのような処理がなされるべきかを思い描いてみて下さい。以下にフローチャートにして掲げます。

図4-6

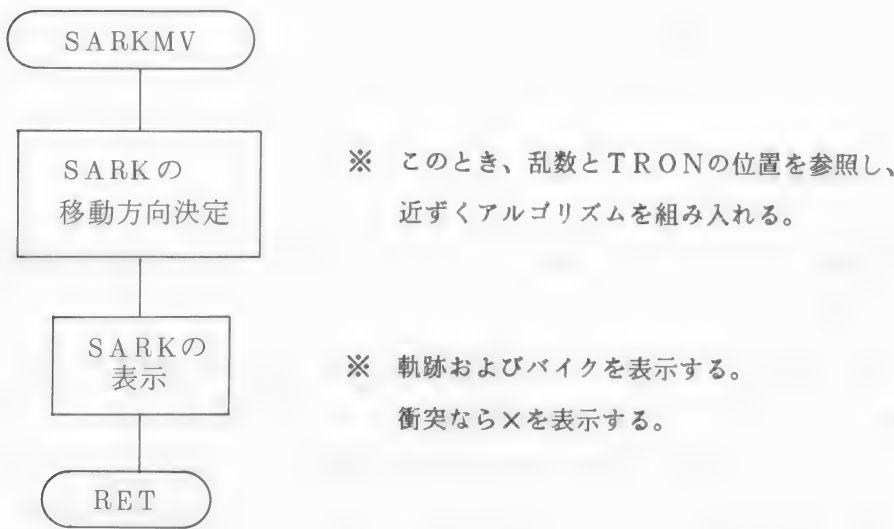


図4-7

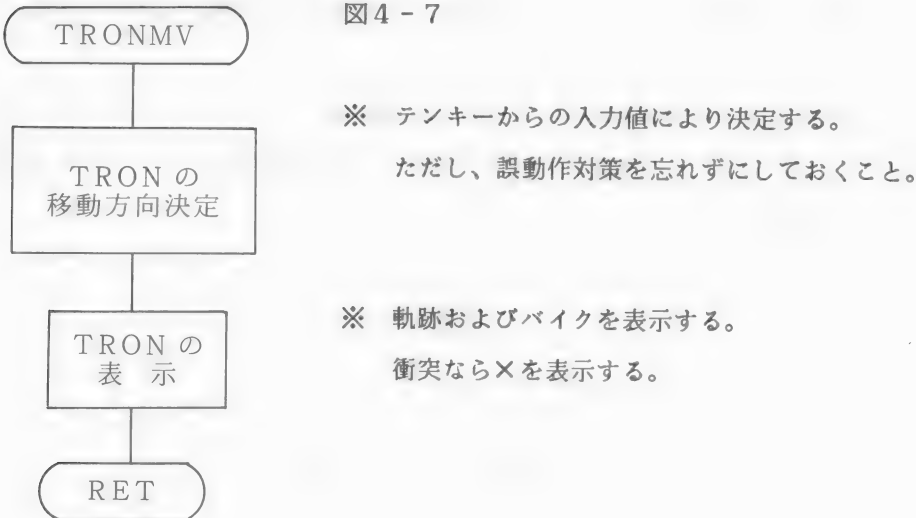
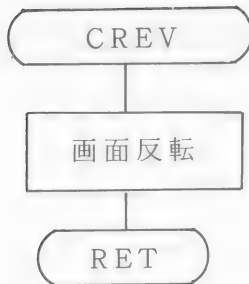


図4 - 8



※ すでに完成したサブルーチンを利用する。

第1段階としては、このような粗いフローチャートで考え、少しずつ詳細を具体化していくことにします。

このような粗いフローチャートでも留意点を取り上げるのに役に立ちます。まず、「SARKの表示」と「TRONの表示」は共通処理が多いことに気づきます。従って、この部分は共通ルーチンとして組み、TRONかSARKかの識別コード（TSフラグとよぶことにします。TS=0はTRON，TS=1はSARKとします。）により内容を変えることにしましょう。この考えは、オールBASIC版でも利用しましたね。TSフラグの「フラグ」は、レジスタのことではなく、メモリー上に設けられた特別な「変数」にあたるものと考えて下さい。

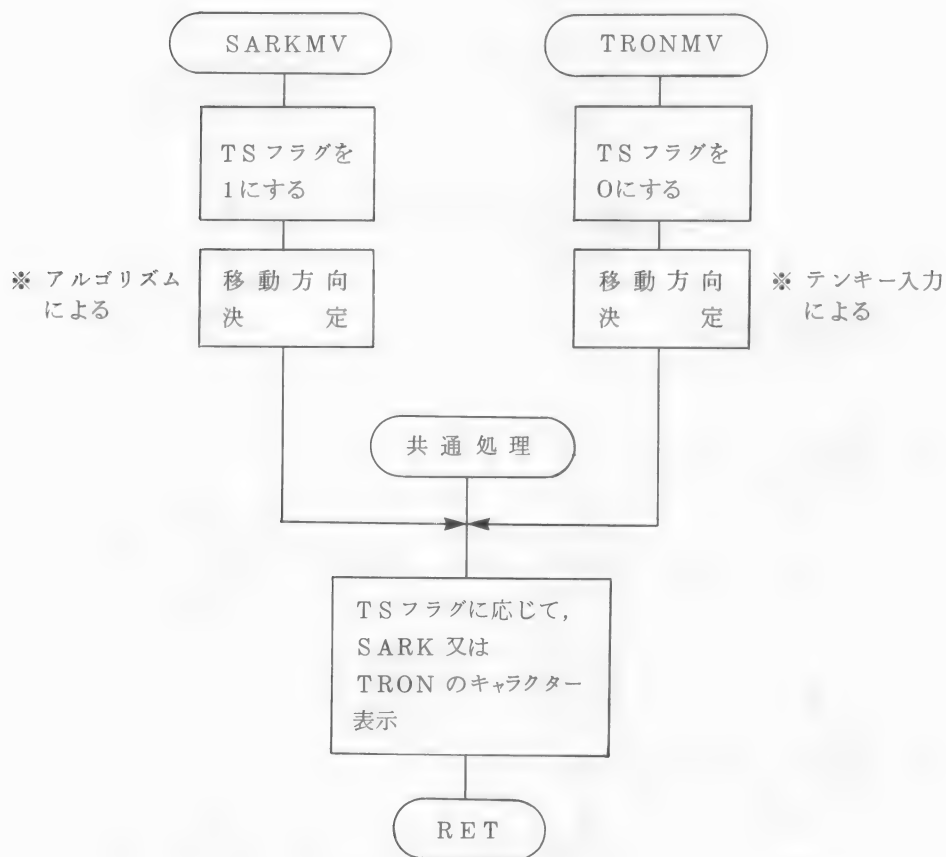
また、次のことにも気づきます。TRONの移動は、テンキー入力（2，4，6，8）によりなされるのだから、これらのテンキー値をもって「移動方向コード」にする方が無駄がない！と。すなわち、

| 移動方向コード | |
|---------|-------|
| 8 = 上 | 2 = 下 |
| 4 = 左 | 6 = 右 |

と決定します（この点はオールBASIC版とは異なります）。

こうして、SARKMV と TRONMV という2つのルーチンは、次のように関連し合うことになりました。

図4-9



以上で、第1段階の作業が終了しました。だんだんイメージが明確になってきましたね。
（実は、オールBASIC版もこのような構成になっているのです。）

4-11 / 移動方向決定ルーチンの具体化

次に移動方向決定のルーチンを具体化してゆきましょう。オールBASIC版を参考に、フローチャートにします。

図4-10

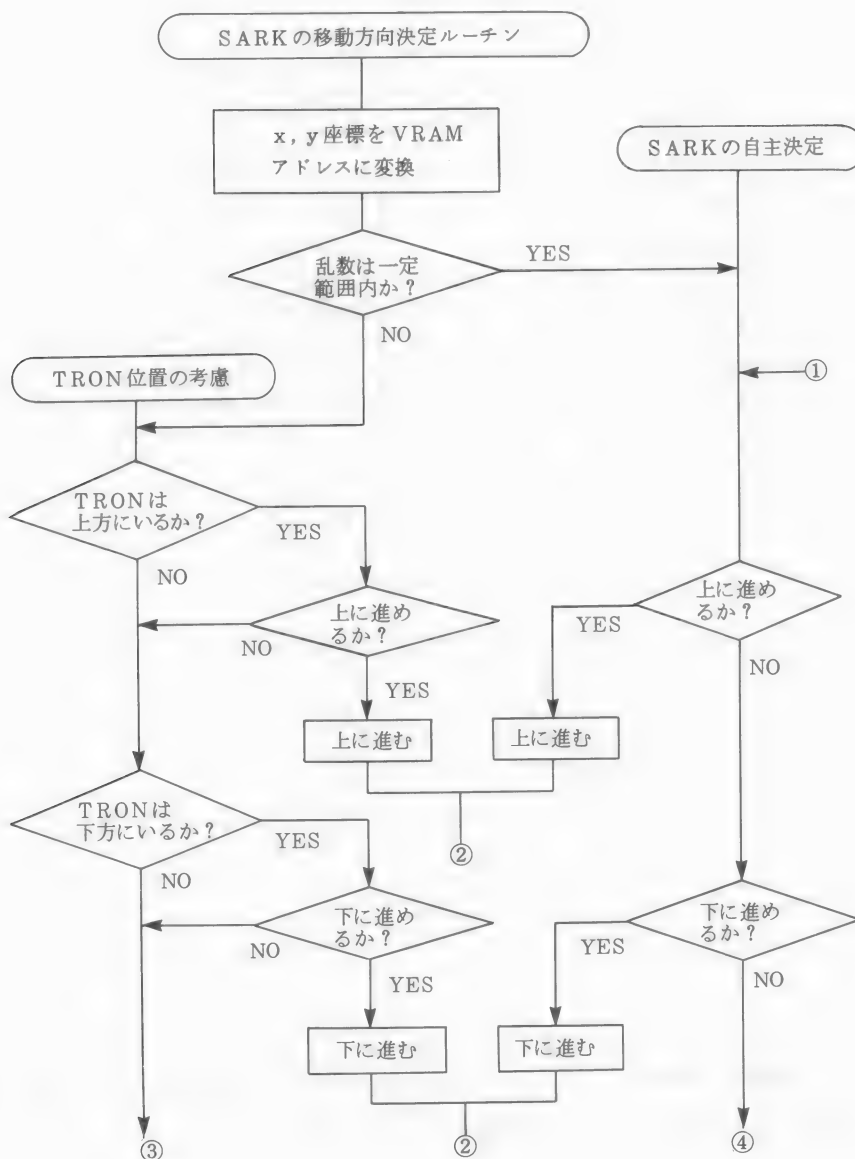


図4-10 続き

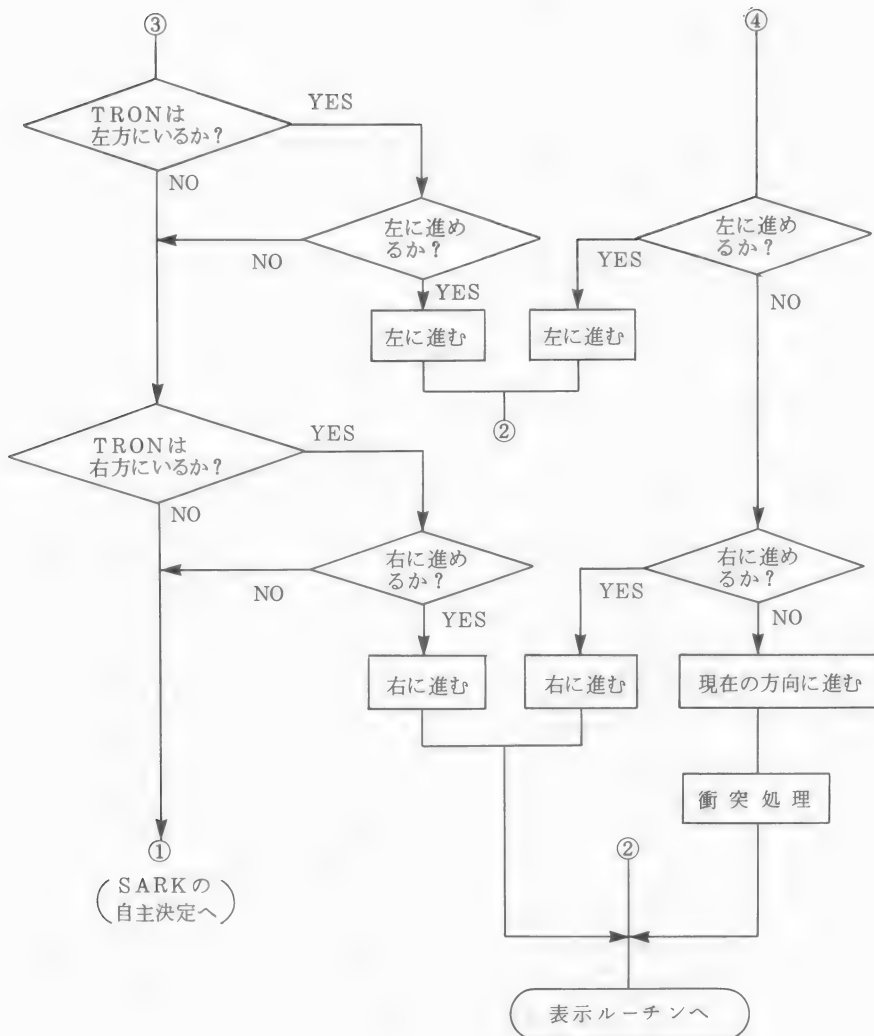
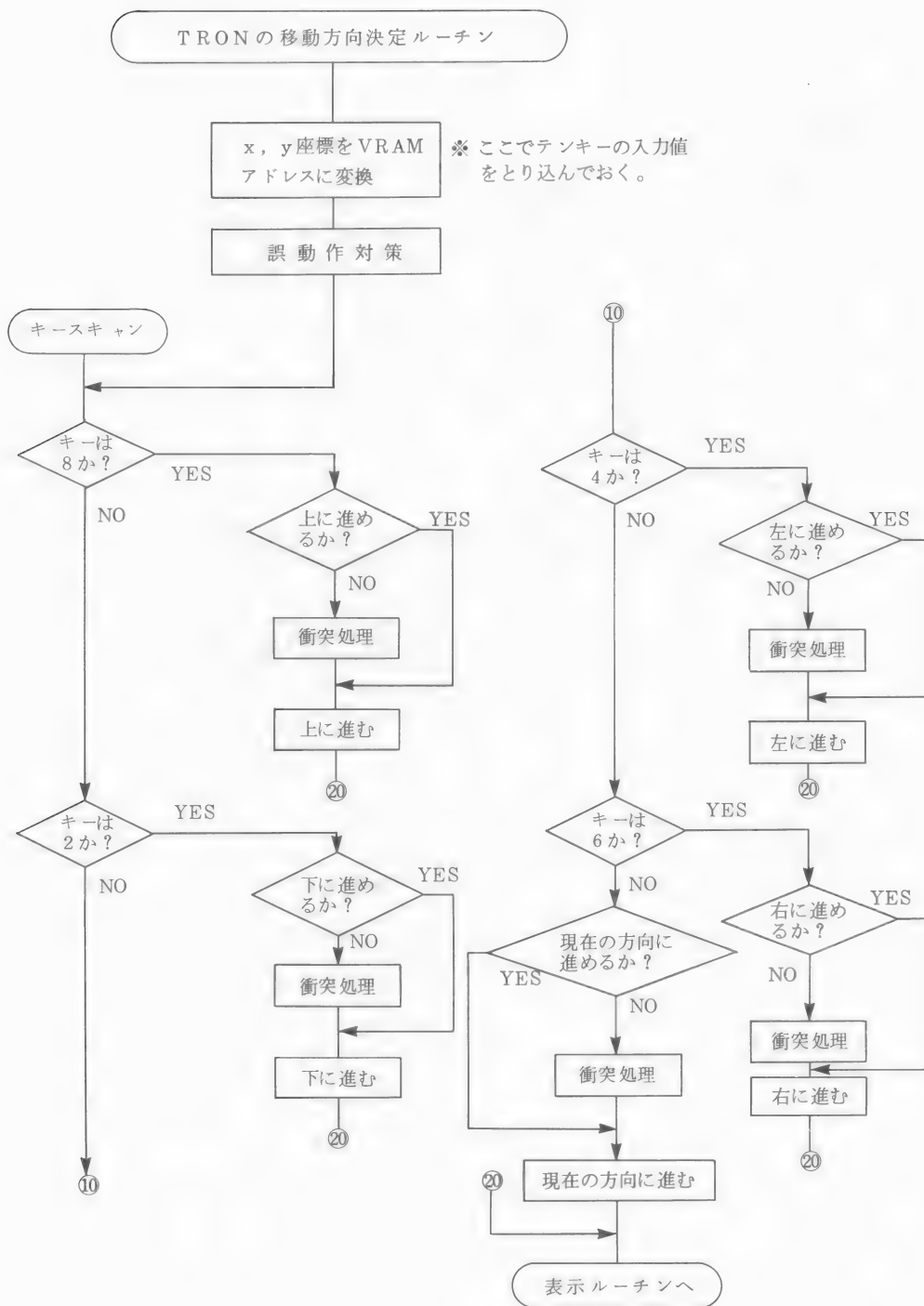


図4-11



以上のフローチャートにより、これらのルーチンから呼び出す下位のサブルーチンが浮かび上がってきます。いずれも、TRON、SARK共用といたします。

表4-1 下位サブルーチン

| ラベル(名称) | 主 要 な 内 容 |
|---------|--|
| ADCAL | x, y座標をVRAMアドレスに変換する。 |
| GOUP | 上に進めるか否かを判断し、進めるならCyフラグを1にして上に進む。進めなければCyフラグを0にし、何もしない。[注] |
| GOUP1 | 無条件に上に進む。 |
| GODN | 下に進めるか否かを判断し、進めるならCyフラグを1にして下に進む。進めなければCyフラグを0にし、何もしない。 |
| GODN1 | 無条件に下に進む。 |
| GOLT | 左に進めるか否かを判断し、進めるならCyフラグを1にして左に進む。進めなければCyフラグを0にし、何もしない。 |
| GOLT1 | 無条件に左に進む。 |
| GORT | 右に進めるか否かを判断し、進めるならCyフラグを1にして右に進む。進めなければCyフラグを0にし、何もしない。 |
| GORT1 | 無条件に右に進む。 |

[注] GOUP, GODN, GOLT, GORT の4つの条件判断つきルーチンでは、判断の結果、実際に進むのか、それとも進まないのかを、サブルーチンから戻った時に判定できなければなりません。そのため、ここでは、Cyフラグ(キャリーフラグ)を用いてみました。Cy=1 なら、実際に進む処理をしたことを、また、Cy=0 なら、進む処理をしなかったことを意味します。

キャリーフラグを1にするには、次のマシン語を用います。

解 説

ニーモニック: SCF [Set Carry Flagの略]

マシンコード: 37

機 能: Cyフラグをセットする。

では逆に、キャリーフラグを0にするには? RESCF のような命令があればよいので

すが、残念ながら「Z80命令表」にはありません。しかし、「キャリーフラグを逆転する命令」 CCF があるので、 SCF の後に、 CCF をすれば、結果として、Cyは0になります。

ここでは別の方法でCyを0にしてみます。論理演算の利用です。

解 説

ニーモニク： OR A

マシンコード： B7

機能： AレジスタとAレジスタの論理和（OR）をとり、Aレジスタに格納する。

次のフラグ変化をする。

●Cyフラグ：つねにリセットされる。

●Zフラグ：Aレジスタの内容が0ならセットされ、0以外ならリセットされる。

論理和というところしげですが、要するに対応するビット毎に次の規則を適用するだけです。

《論理和の規則》

1, 1 ⇒ 1

1, 0 ⇒ 1

0, 1 ⇒ 1

0, 0 ⇒ 0

たとえば、10101011（2進表示）と、11001100（2進表示）の論理和をとると、

| | | |
|------|----------|--------|
| | 10101011 | |
| 論理和) | 11001100 | |
| | <hr/> | |
| | 11101111 | (2進表示) |

となります。さて、OR A という時には、AレジスタとAレジスタの論理和で、どのビットも同じですから、

| |
|----------------|
| 1, 1⇒1, 0, 0⇒0 |
|----------------|

の規則から、Aレジスタの内容は不変です。ただし、フラグの変化が重要です。ですから、この命令は、「Cyフラグのリセット」や、「Aレジスタの内容が0か否かを判定する」のによく用いられます。

4-12/キャラクター表示ルーチンの具体化

最後に、TRON、SARK共用のキャラクター表示ルーチンを具体化しておきましょう。キャラクターのASCIIコードを確認しておきます（ユーザー定義文字として使用）。

表4-2

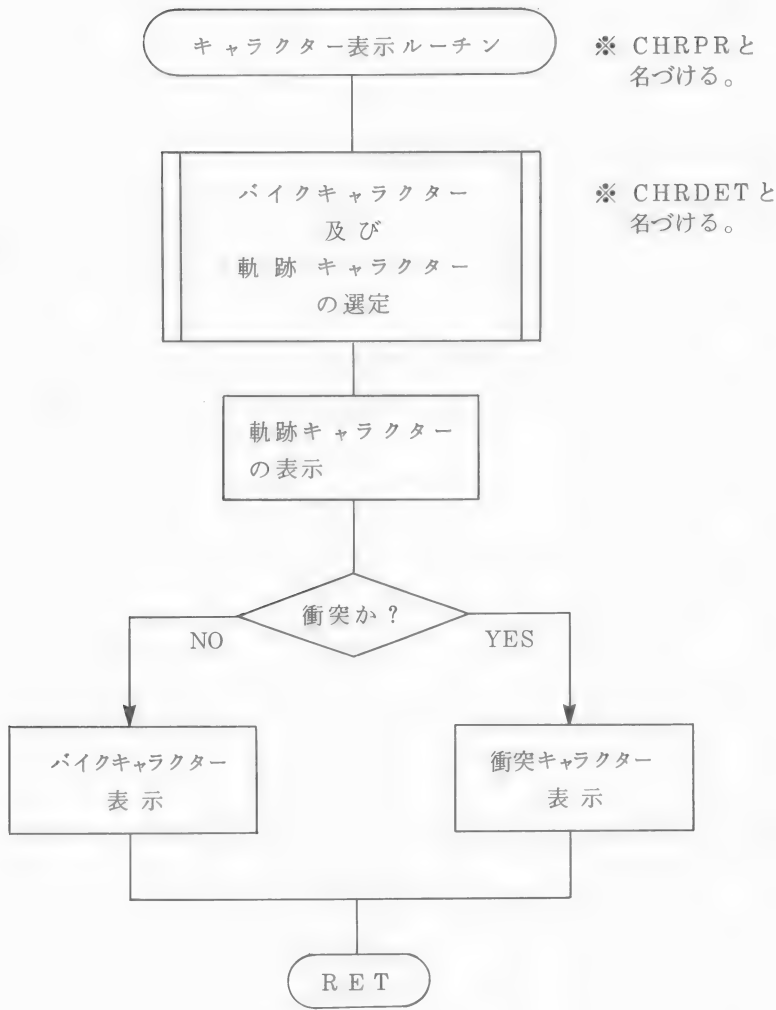
| バイト・キャラクター | | ASCIIコード (10進) | |
|------------|-----|----------------|-----|
| TRON | 上向き | 64H | 100 |
| | 下向き | 6EH | 110 |
| | 左向き | 78H | 120 |
| | 右向き | 82H | 130 |
| SARK | 上向き | C8H | 200 |
| | 下向き | D2H | 210 |
| | 左向き | DCH | 220 |
| | 右向き | E6H | 230 |

| 軌跡キャラクター | ASCIIコード |
|----------|----------|
| — | 90H |
| | 91H |
| ┐ | 97H |
| └ | 98H |
| ┐ | 99H |
| └ | 9AH |

| 衝突キャラクター (標準モード) | ASCIIコード |
|------------------|----------|
| × | E8H |

フローチャートは以下の通りです。

図4-12



キャラクター選定サブルーチン（CHRDET）は、次のように具体化します。

4-13/ワークエリアの設定

BASICでは、「変数」が使用できるので、私たちは変数により、保存すべきデータの管理をBASICインタプリタに任せることができました。しかし、自前で作成するマシン語プログラムでは、この作業も自分で行なわねばなりません。すなわち、各ルーチンの処理で保存すべきデータは、メモリーのどこかに格納しておく必要があります。このような目的のために設けられるメモリー上のエリアをワークエリア (working area) と申します。

前節までに具体化したフローチャートにより、私たちは、各サブルーチンで必要とするデータが何であるかを明確にすることができます。

SARKの移動方向を決定するアルゴリズムのためには、TRON, SARKのx, y座標が各々必要ですね。また、軌跡キャラクターの表示で、TRON, SARKが現在進んでいる方向を参照しますので、各々の移動方向コードも保存すべきデータです。共用ルーチン内で処理を分けるためには、TRON, SARKの識別コードが必要です。SARKの行動に偶然性を加味するには、乱数を使いました。衝突かどうかの判定コードも必要でしたね。

サブルーチン ADCAL では、座標をVRAMアドレスに変換しますが、結果を格納する場所がいります。移動の下位ルーチンでは、次に進む位置のVRAMアドレスも計算されるはずですね。

また、表示ルーチンでは、軌跡・バイクのASCIIコードを格納する場所が必要です。

以上から、次のように設定します。



このように各部分をメモリー上のどこに配置するかを示した図を メモリーマップ とよぶのでしたね。マシン語プログラムを作成する時は、必ずメモリーマップを明確にしておきましょう。

またワークエリアの詳細は次のように決めます。

表4-3 ワークエリア

| アドレス | 内 容 | ラベル (名称) |
|-------|--------------------------------|----------|
| E000H | TRON x座標 | TRX |
| E001H | TRON y座標 | TRY |
| E002H | SARK x座標 | SKX |
| E003H | SARK y座標 | SKY |
| E004H | TRON 移動方向コード (2, 4, 6, 8) | TRMV |
| E005H | SARK 移動方向コード | SKMV |
| E006H | 乱数 | RND |
| E007H | 続行フラグ (0=続行, 1=終了) | FLAG |
| E008H | VRAM 現アドレス 下位バイト | ADR |
| E009H | VRAM 現アドレス 上位バイト | |
| E00AH | VRAM 新アドレス 下位バイト | NADR |
| E00BH | VRAM 新アドレス 上位バイト | |
| E00CH | (計算用) 移動方向コード | MOVE |
| E00DH | TRON, SARKフラグ (0=TRON, 1=SARK) | TS |
| E00EH | 軌跡キャラクターのASCIIコード | ORBIT |
| E00FH | バイクキャラクターのASCIIコード | BIKE |

ワークエリアの各アドレスには、引用しやすいように、ラベル (名前) をつけておきます。
(アドレスのように2バイトの場合は、上下位逆転が起きています。この時は、下位バイトの方のみにラベルをつけました。)

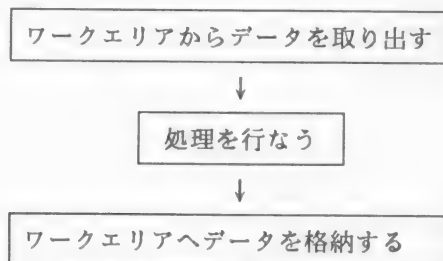
4-14/データ引き渡しの仕様

BASICの「変数」にあたるワークエリアの仕様がはっきりしましたので、次に各ルーチンが、どのようにワークエリアを書き換えてゆくかを明確に決めます。

この問題は、サブルーチン間のデータ引き渡しについての取り決めとも言えます。BASICインタプリタ内には、多くのマシン語サブルーチンがありますが、これらの間でのデータ引き渡しは、レジスタを通じて行なわれることが多いようです。この方法は、省メモリー・高速化を図れますが、レジスタは数に限りがあるため、CPUの動作に伴い頻繁に値が変わります。従って、このような方法でシステムを組む時は、各サブルーチン内でのレジスタの値の動きを追跡し、保存されるレジスタ、書き換えられるレジスタをきちんと把んでおかねばなりま

せん。これは、初心者にとってはなかなか大変な仕事なのです（私もそうでした）。

そこで本書のゲーム作成では、速度が多少遅くなり、メモリーが余計に使われるのをガマンしてでも、特定のデータを格納しておくメモリーのエリア（ワークエリア）を経由して、サブルーチン間のデータ引き渡しを行なう方法をとることにします。これにより、各サブルーチンでのデータの動きは次図のようになり、単純でわかりやすいプログラムにできます。



以下、ワークエリアの動きをサブルーチン毎に決めてゆきます。〔アドレスはラベルで記します。〕

表4-4 《下位サブルーチン》

| 名 称 | 参照するアドレス | 書き換えるアドレス |
|-----------------|---|--|
| ADCAL | TS トロンのとき テンキー入力 (HLレジスタペアの示すアドレス) TRX, TRY サークのとき SKMV SKX, SKY | MOVE ADR (2バイト) |
| GOUP1 (上へ進む) | ADR (2バイト) TS トロンのとき、TRY サークのとき、SKY | NADR (2バイト) MOVE (←8) TRY (1減ずる) SKY (1減ずる) |

| | | |
|--------------------------|--|--|
| GODN1 (下へ進む) | ADR (2バイト) TS トロンのとき、TRY サークのとき、SKY | NADR (2バイト) MOVE (←2) TRY (1増やす) SKY (1増やす) |
| GOLT1 (左へ進む) | ADR (2バイト) TS トロンのとき、TRX サークのとき、SKX | NADR (2バイト) MOVE (←4) TRX (1減ずる) SKX (1減ずる) |
| GORT1 (右へ進む) | ADR (2バイト) TS トロンのとき、TRX サークのとき、SKX | NADR (2バイト) MOVE (←6) TRX (1増やす) SKX (1増やす) |
| CHRDET (キャラクター 選定) | MOVE TS トロンのとき、TRMV サークのとき、SKMV | ORBIT BIKE トロンのとき、TRMV サークのとき、SKMV |

表4-5 《上位サブルーチン》

| 名 称 | 参照するアドレス | 書き換えるアドレス |
|--------|--------------------------------------|-------------------------------|
| SARKMV | RND TRY, SKY, TRX, SKX MOVE | TS (←1) 衝突のとき FLAG (←1) |
| TRONMV | MOVE | TS (←0) 衝突のとき FLAG (←1) |

以上で、サブルーチンの仕様が決まりました。実行手順を考えて、データの動きを追ってみましょう。

図 4 - 1 4 《データの動き》

(⇒ は書き換わるアドレス)

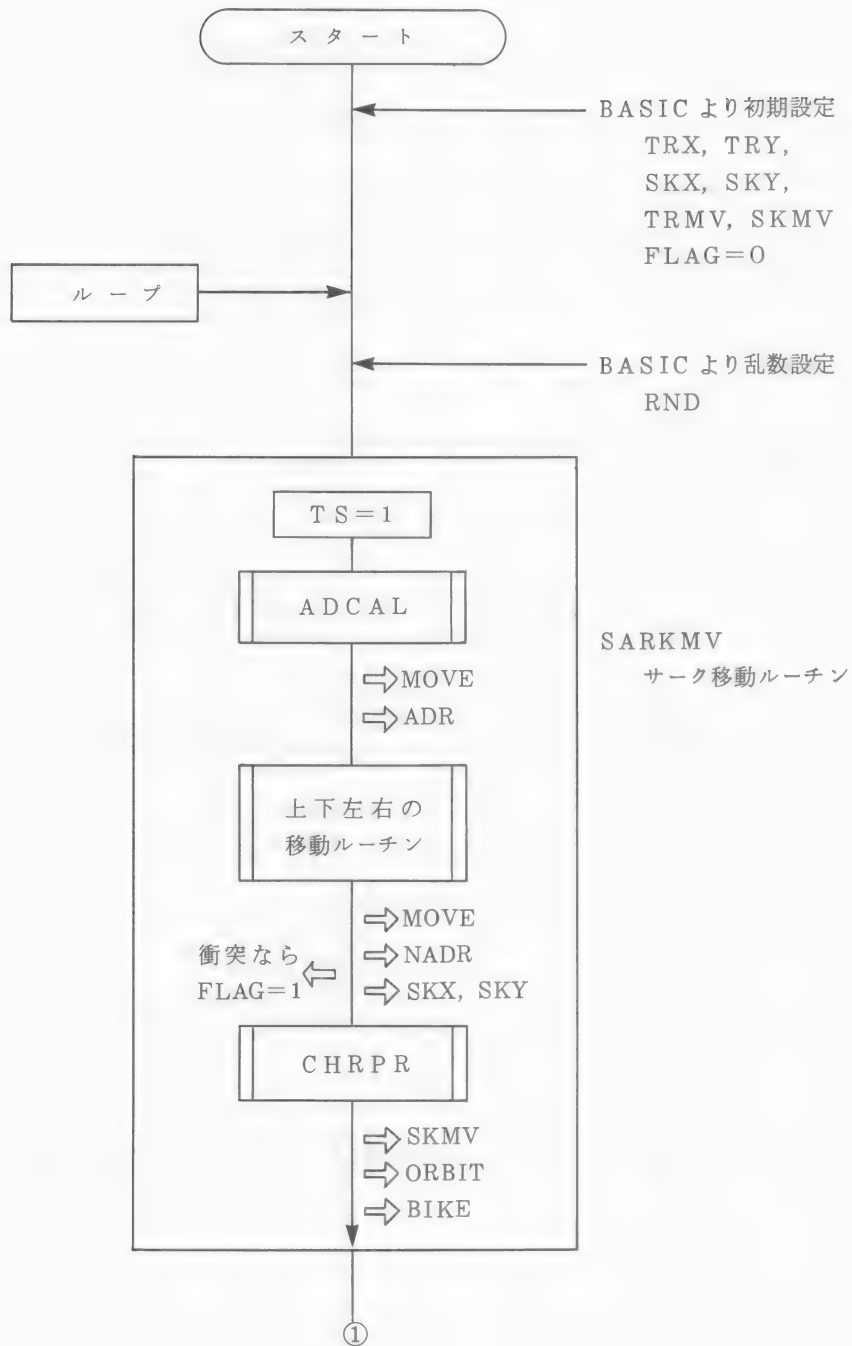
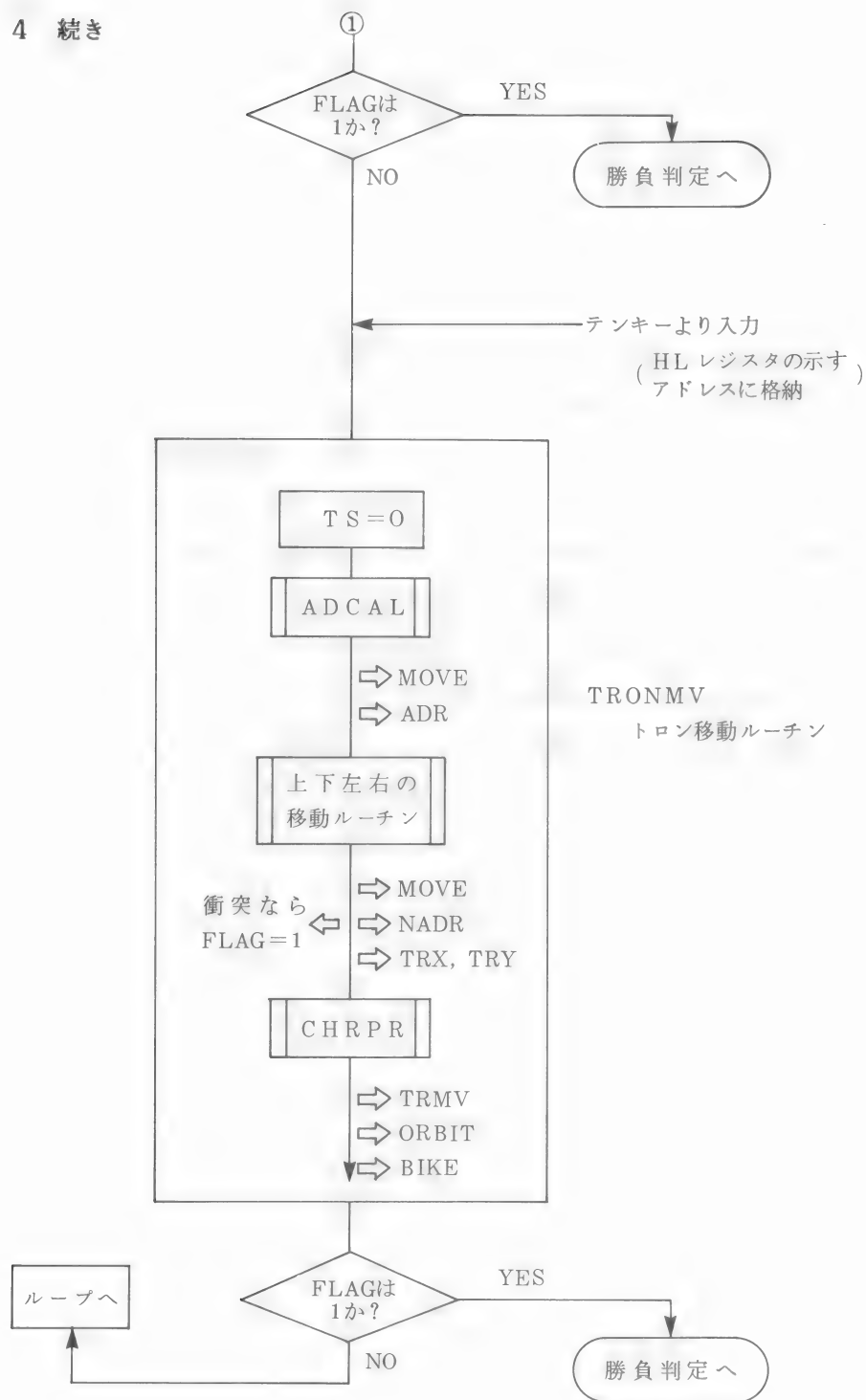


図4-14 続き



この図と、各ルーチンの参照アドレスの表を見れば、各ルーチンが、データを受けとり、書き換え、それが次のルーチンの初期値となり…という過程が理解できます。こうして、作成してゆくサブルーチンの仕様の詳細も決まりました。あとは —— そう、マシン語でプログラムを書き下してゆくだけです。

□ コーヒー・ブレイク

久々にコーヒータイムです。マシン語プログラムを組むまで随分準備がいりました。BASICでしたら、対話機能により、いきなり画面に向かってプログラムを作り始めても、エラー表示などでシステムが教えてくれました。しかし、マシン語では、エラーや設計ミスがあると、即「暴走」してしまいます。そこで、事前に入念な設計図が必要なのです。

本当は、プログラムを書き下す前に、さらにもう1段階の作業 —— マシン語と対応できる詳細なフローチャートを書く —— をすると理想的ですが、ページ数が余りに多くなりそうなので、本書では便宜的にこの段階を省くことにします。

皆様が自身で、マシン語プログラムを作られる時には、くれぐれも入念な準備をお忘れなく。この方が結果として、開発時間が少なくて済みますよ！

また本書のような作り方（大ざっぱに決めてから次第に詳細化してゆく）をトップダウンの方法とよびますが、私自身はこのような方法の方が、仕様の変更などにも早く対応できて好きです。これに対して、最下位のサブルーチンから作り、これらを組み上げて大きなプログラムにしていく方法をボトムアップの方法とよんでいます。

4-15／サブルーチンSARKMVの作成

まず、サークの移動方向決定サブルーチンである SARKMV をマシン語で書き下してみしましょう。(紙に書いてもよいし、BASICのスクリーンエディット機能を用いて、REM文などで、BASICテキストとして書いていってもよいと思います。)

おおよそ、リスト (図4-15) のようになるはずです。まず、ニーモニックを書いてゆきます。サブルーチンやジャンプ先には適当にラベルをつけて記します。次に、それらを、マシンコードに直し(ハンドアセンブル!)、アドレスを割りふってゆきます。コメントは読者の理解のためにつけましたから、書かなくて構いません。

サブルーチンやジャンプ先のアドレスが未定のうちは、必要なバイト数だけ線を引き、場所をあけておきます。

ルーチンの終わりまで書ききったら、わかる部分について、絶対アドレス、相対アドレスを書き入れてゆきます。たとえば、SELFとラベルをつけたアドレスは、D044Hとわかりますね。それから、D061H番地の相対ジャンプのジャンプ先は、D068H番地になりましたから、D063Hを基点0と数えて、5番地先、すなわち相対アドレスは05Hとなります。もう1つやってみますか。

D066H番地の相対ジャンプのジャンプ先は、D07DH番地です。D068Hを基点0と数えて、21番地先、すなわちこの相対アドレスは15Hとなります。相対アドレスの計算法よろしいですか。残りについても、計算して記入して下さい。

図4 - 15

■ SARKMV (SARK ノ イトウ ルーチン) ■

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|---------------------|-------------------------------------|
| D000 | 3E01 | LD A, 01H | |
| D002 | 320DE0 | LD (0E00DH), A | ;TS ← 1 |
| D005 | CD | CALL ADCAL | ;アドレス クイザン |
| D008 | 3A06E0 | LD A, (0E006H) | |
| D00B | FE04 | CP 04H | ;ランズワ ヲ 4 ト、クラハベル |
| D00D | DA | JP C, SELF | ;ランズワ < 4 フラ、SARK ノ シュケツタイ ヲ JUMP |
| D010 | 3A01E0 | LD A, (0E001H) | ;TRON ノ イチ ヲ ミル アルゴリズム |
| D013 | 2103E0 | LD HL, 0E003H | |
| D016 | BE | CP (HL) | ;TRY ト SKY ヲ クラハベル |
| D017 | DC | CALL C, GOUP | ;TRY < SKY フラ、GOUP ヲ |
| D01A | DA | JP C, CHRPR | ;GOUP ショリ ヲ シタラ、CHRPR ルーチン ヲ JUMP |
| D01D | 3A01E0 | LD A, (0E001H) | |
| D020 | 2103E0 | LD HL, 0E003H | |
| D023 | BE | CP (HL) | ;TRY ト SKY ヲ クラハベル |
| D024 | C4 | CALL NZ, GODN | ;TRY > SKY フラ、GODN ヲ |
| D027 | DA | JP C, CHRPR | ;GODN ショリ ヲ シタラ、CHRPR ルーチン ヲ JUMP |
| D02A | 3A00E0 | LD A, (0E000H) | |
| D02D | 2102E0 | LD HL, 0E002H | |
| D030 | BE | CP (HL) | ;TRX ト SKX ヲ クラハベル |
| D031 | DC | CALL C, GOLT | ;TRX < SKX フラ、GOLT ヲ |
| D034 | DA | JP C, CHRPR | ;GOLT ショリ ヲ シタラ、CHRPR ルーチン ヲ JUMP |
| D037 | 3A00E0 | LD A, (0E000H) | |
| D03A | 2102E0 | LD HL, 0E002H | |
| D03D | BE | CP (HL) | ;TRX ト SKX ヲ クラハベル |
| D03E | C4 | CALL NZ, GORT | ;TRX > SKX フラ、GORT ヲ |
| D041 | DA | JP C, CHRPR | ;GORT ショリ ヲ シタラ、CHRPR ルーチン ヲ JUMP |
| D044 | CD | CALL GOUP | ;SELF (SARK ノ シュケツタイ ルーチン) |
| D047 | DA | JP C, CHRPR | |
| D04A | CD | CALL GODN | |
| D04D | DA | JP C, CHRPR | |
| D050 | CD | CALL GOLT | |
| D053 | DA | JP C, CHRPR | |
| D056 | CD | CALL GORT | |
| D059 | DA | JP C, CHRPR | |
| | | | ; → イカ ハ、スズメル ホウコウ カ、ナイトキ ノ ショリ |
| D05C | 3A0CE0 | LD A, (0E00CH) | ;SARK ショフトツ ルーチン ※※ E00CH = MOVE ※※ |
| D05F | FE08 | CP 08H | ;ホウコウ ハ ウエ カ ? |
| D061 | 20 | JR NZ, シタ ト ヒカク ヲ | ;フエ テアケレハ、シタ ト ヒカク ヲ |
| D063 | CD | CALL GOUP1 | ;フエ ニ スズム |
| D066 | 18 | JR SARK ショフトツ ヲ | |
| D068 | FE02 | CP 02H | ;ホウコウ ハ シタ カ ? |
| D06A | 20 | JR NZ, ヒタバリ ト ヒカク ヲ | ;シタ テアケレハ、ヒタバリ ト ヒカク ヲ |
| D06C | CD | CALL GODN1 | ;シタ ニ スズム |
| D06F | 18 | JR SARK ショフトツ ヲ | |
| D071 | FE04 | CP 04H | ;ホウコウ ハ ヒタバリ カ ? |
| D073 | 20 | JR NZ, ミキ ト ヒカク ヲ | ;ヒタバリ テアケレハ、ミキ ト ヒカク ヲ |
| D075 | CD | CALL GOLT1 | ;ヒタバリ ニ スズム |
| D078 | 18 | JR SARK ショフトツ ヲ | |
| D07A | CD | CALL GORT1 | ;ミキ ニ スズム |
| D07D | 3E01 | LD A, 01H | ;SARK ショフトツ ! |
| D07F | 3207E0 | LD (0E007H), A | ;FLAG ← 1 |
| D082 | C3 | JP CHRPR | ;ヒョウシ ルーチン ヲ JUMP |

4-16/サブルーチンTRONMVの作成

次に、トロンの移動ルーチン TRONMV をマシン語にしましょう。

リスト (図4-16) のようにしてみました。

移動方向を決める部分には、2つの入り口を作りました。1つは、テンキーの入力値をもって、D09BH番地へ入る場合、もう1つはD098H番地から入る場合です。後者は、テンキーの値ではなく、現在トロンの進んでいる方向コード (TRMVに格納されている) をもって、D09BH番地へ行きます。これは「現状維持」の場合で、テンキー値が、2, 4, 6, 8 のいずれでもない時は、D0DAH番地から、ここへジャンプするようにしました。ですから、D0DAH番地の相対ジャンプの相対アドレスは、D0DCH番地を基点0として、68番地手前となりますから、-68を符号つき16進数で表わして、相対アドレスはBCHとなりますね。

SARKMVの時と同様に、この段階でわかる絶対アドレス、相対アドレスを記入して下さい。

図4-16

TRONMV (TRON ノ イトバ ルーチン)

| アドレス | マシンコード | メモニック | コメント |
|------|--------|-------------------|---|
| D085 | 3E00 | LD A, 00H | |
| D087 | 320DE0 | LD (0E00DH), A | ;TS ← 0 |
| D08A | CD | CALL ADCAL | ;アドレス タイプ |
| D08D | 3A04E0 | LD A, (0E004H) | ;コトバウ タイプ *** E004H = TRMV *** |
| D090 | 210CE0 | LD HL, 0E00CH | ; *** E00CH = MOVE *** |
| D093 | 86 | ADD A, (HL) | ;A ← A + (HL) |
| D094 | FE0A | CP 0AH | ;A 10 ト クラベル (TRMV ト MOVE カ ハンタイ ホウゴウ ナラ、10 ニ フル !) |
| D096 | 20 | JR NZ, キー・スキャン ↑ | ;ハンタイ テンキー・スキャン ↑ |
| D098 | 2104E0 | LD HL, 0E004H | ;テンキー・スキャン ↑ |
| D09B | 7E | LD A, (HL) | ; *** キー・スキャン *** *** HL = テンキー ノ データ ノ アル アドレス *** |
| D09C | FE08 | CP 08H | ;8 ト クラベル |
| D09E | 20 | JR NZ, 2 トノ ヒカク ↑ | |
| D0A0 | CD | CALL Goup | ;UI ニ スズメル ? |
| D0A3 | 38 | JR C, EXIT8 | ;Goup ショリ ヲ シタラ、EXIT8 ↑ |
| D0A5 | CD | CALL Goup1 | ;UI ニ スズル (ショウトツ !) |
| D0A8 | 3E01 | LD A, 01H | |
| D0AA | 3207E0 | LD (0E007H), A | ;FLAG ← 1 (ショウトツ) |
| D0AD | C3 | JP CHRPR | ;EXIT8 (8 ノ ルーチン ノ テクニク) |
| D0B0 | FE02 | CP 02H | ;2 ト クラベル |
| D0B2 | 20 | JR NZ, 4 トノ ヒカク ↑ | |

図4-16 続き

| | | | |
|------|---------|-------------------|--------------------------|
| D0B4 | CD ____ | CALL GODN | ;シタ ニ スズメル ? |
| D0B7 | 38 ____ | JR C,EXIT2 | ;GODN ショリ ヲ シタラ、EXIT2 ヲ |
| D0B9 | CD ____ | CALL GODN1 | ;シタ ニ スズメル (ショウトツ !) |
| D0BC | 3E01 | LD A,01H | |
| D0BE | 3207E0 | LD (0E007H),A | ;FLAG <— 1 (ショウトツ) |
| D0C1 | C3 ____ | JP CHRPR | ;EXIT2 |
| D0C4 | FE04 | CP 04H | ;4 ト クラハメル |
| D0C6 | 20 ____ | JR NZ, 6 トノ ヒカク ヲ | |
| D0C8 | CD ____ | CALL GOLT | ;ヒタマリ ニ スズメル ? |
| D0CB | 38 ____ | JR C,EXIT4 | ;GOLT ショリ ヲ シタラ、EXIT4 ヲ |
| D0CD | CD ____ | CALL GOLT1 | ;ヒタマリ ニ スズメル (ショウトツ !) |
| D0D0 | 3E01 | LD A,01H | |
| D0D2 | 3207E0 | LD (0E007H),A | ;FLAG <— 1 (ショウトツ) |
| D0D5 | C3 ____ | JP CHRPR | ;EXIT4 |
| D0D8 | FE06 | CP 06H | ;6 ト クラハメル |
| D0DA | 20 ____ | JR NZ, ケンショウ イシ ヲ | ; (アドレス = D098H) |
| D0DC | CD ____ | CALL GORT | ;ミキニ スズメル ? |
| D0DF | 38 ____ | JR C,EXIT6 | ;GORT ショリ ヲ シタラ、EXIT6 ヲ |
| D0E1 | CD ____ | CALL GORT1 | ;ミキニ スズメル (ショウトツ !) |
| D0E4 | 3E01 | LD A,01H | |
| D0E6 | 3207E0 | LD (0E007H),A | ;FLAG <— 1 (ショウトツ) |
| D0E9 | C3 ____ | JP CHRPR | ;EXIT6 |

4-17/移動用下位ルーチンの作成

TRONMV の後には、移動に使われる（すなわち、SARKMV、TRONMVから呼び出される）下位サブルーチンを配置することにします。

まず、アドレス計算ルーチン —— ADCAL —— を置きましょう。トロノ、サーク共用であることに注意して、書き下すとリスト（図4-17）のようになるでしょう。

新しく出てきた命令がありますから、説明を与えておきます。

解 説

ニーモニック： DJNZ e （eは符号つき1バイト数値）

マシンコード： 10 e

機能： Bレジスタの値から1を引き、0にならなければ相対アドレス e 番地へジャンプ。Bレジスタの値が0になったら次へ進む。

ニーモニクの DJNZ は、Decrement Jump Non Zero の略で、Bレジスタをカウンタとして自動的に相対ジャンプしてくれるので、255回までのループ処理には便利な命令です。

D114H番地にこの命令を書きましたが、相対アドレスの計算は、もう慣れましたか？ ジャンプ先は D113H ですから、D116H を基点0として、3番地手前となり、-3 を符号つき16進表示して FDH が相対アドレスですね。

ところで、このループは何をしているかわかりですか？ HLレジスタペアの初期値は 3000H です。また、DEレジスタペアには、(トロンまたはサークの) y座標が入っています。HL ← HL+DE を40回くり返すのですから、結局、HLレジスタペアには $3000H + y * 40$ という計算の答が入ります。

これに、ADD HL, BC を行なうと、Bレジスタは今は0であることに注意し、BCにはx座標が入っている訳ですから、HLレジスタペアには $3000H + y * 40 + x$ すなわち、VRAMのアドレスが入ることになるのですね。

このように、「かけ算」の1つの方法は、たし算のくり返しで実現できることを覚えておきましょう。

また説明を特にしませんでしたでしたが、16ビットの加算命令

ADD HL, DE

ADD HL, BC

についてはよいですね。Aレジスタが、8ビットのアキュムレータであったのと同様に、HLレジスタペアは、16ビットのアキュムレータとしての役割も持っていることに注意しましょう。ですから、16ビットの計算では普通、「HLレジスタペアに対して加減を行ない、結果をHLに格納する」という形をとることを覚えておいて下さい。

また、D117H番地の LD (0E008H), HL もよろしいですか？ Z80の鉄則で、上下位逆転が起こって、

E008H番地 ← Lレジスタの値が入る
(VRAMのアドレス下位バイト)

E009H番地 ← Hレジスタの値が入る
(VRAMのアドレス上位バイト)

となりますね。

相対アドレスをすべて記入して下さい。

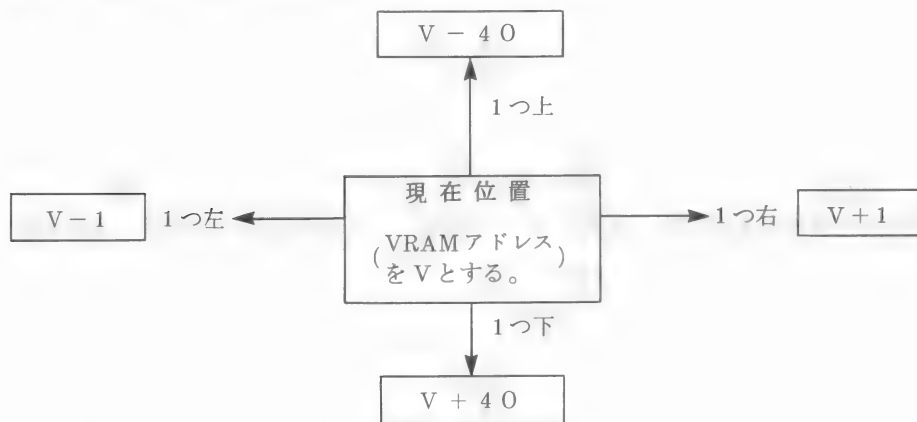
図4-17

■ ADCAL (アドレス ケイサン サブルーチン) ■

| アドレス | マシンコード | メモリアドレス | コメント |
|------|--------|--------------------|---|
| D0EC | 3A0DE0 | LD A, (0E00DH) | ; ** E00DH = TS |
| D0EF | FE00 | CP 00H | ; トロノ カ ? |
| D0F1 | 20__ | JR NZ, サーク ノ ショリ ヲ | |
| D0F3 | 7E | LD A, (HL) | ; トロノ ノ ショリ ** HL = テンキ- ノ アライ ノ アル アドレス ** |
| D0F4 | 320CE0 | LD (0E00CH), A | ; ** E00CH = MOVE ** |
| D0F7 | 2100E0 | LD HL, 0E000H | ; ** E000H = TRX ** |
| D0FA | 56 | LD D, (HL) | ; D ← TRX ノ アライ |
| D0FB | 23 | INC HL | ; HL ← HL + 1 |
| D0FC | 5E | LD E, (HL) | ; E ← TRY ノ アライ |
| D0FD | 18__ | JR アドレス ケイサン ヲ | |
| D0FF | 3A05E0 | LD A, (0E005H) | ; サーク ノ ショリ ** E005H = SKMV ** |
| D102 | 320CE0 | LD (0E00CH), A | ; ** E00CH = MOVE ** |
| D105 | 2102E0 | LD HL, 0E002H | ; ** E002H = SKX ** |
| D108 | 56 | LD D, (HL) | ; D ← SKX ノ アライ |
| D109 | 23 | INC HL | ; HL ← HL + 1 |
| D10A | 5E | LD E, (HL) | ; E ← SKY ノ アライ |
| D10B | 210030 | LD HL, 3000H | ; アドレス ケイサン ** 3000H = VRAM トップ アドレス ** |
| D10E | 4A | LD C, D | ; D レジスタ ラ C レジスタ ヲ タイヒ サセル |
| D10F | 1600 | LD D, 00H | ; DE = Y サビョウ ノ アライ |
| D111 | 0628 | LD B, 28H | ; B = ル-フ カイズ 40 カイ |
| D113 | 19 | ADD HL, DE | ; HL ← HL + DE |
| D114 | 10__ | DJNZ D113H ヲ | ; B レジスタ ラ カウンター トシテ ル-フ サセル |
| D116 | 07 | ADD HL, BC | ; HL ← 3000H + Y*40 + X ** VRAM ノ アドレス ** |
| D117 | 2208E0 | LD (0E008H), HL | ; ** E008H = ADR ** |
| D11A | C9 | RET | |

続いて、G O U P, G O U P 1, G O D N, G O D N 1, G O L T, G O L T 1, G O R T, G O R T 1 の上下左右移動サブルーチンを書くことにします。

移動とVRAMアドレスについて注意しておきます。



従って、HLレジスタペアに現在位置のVRAMアドレスの値を入れるとき、次により各アドレスを計算できます。

1つ下の位置 : LD DE, 0028H (10進で40)

ADD HL, DE

1つ左の位置 : DEC HL

1つ右の位置 : INC HL

注意すべきなのは、「1つ上の位置」を計算するものです。HL←HL-DE にあたる命令があるとよいのですが、「Z80命令表」にはありません。近いものとしては次があります。

解 説

ニーモニック: SBC HL, DE

マシンコード: ED 52

機能: HL←HL-DE-Cy

すなわち、Cyフラグ込みでの16ビット減算命令です。従って、Cyフラグを0にした後

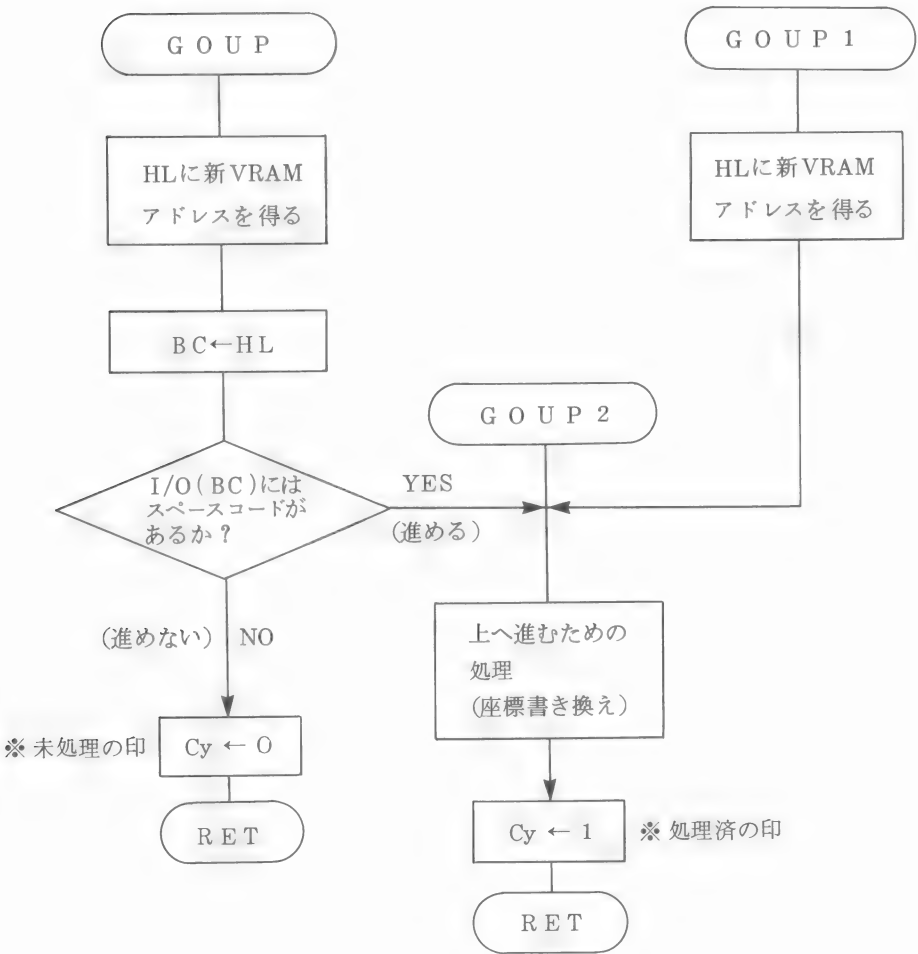
に使用すれば目的を達することができます。

```
1つ上の位置  : LD  DE, 0028H (10進で40)
                OR  A          (Cyをリセット)
                SBC HL, DE
```

Cyフラグをリセットするための論理和 OR A の使い方、よろしいですか？

各移動ルーチンの構成ですが、すべて次のような仕組みになっています。代表としてG O U P, G O U P 1について述べます。他も同様です。

図4-18



次に、各ルーチンのリストを掲げます。相対アドレスを記入して下さい。

図4-19~22

■ GOUP (ハンダツキでUIニスルサブルーチン) ■ (図4-19①)

| アドレス | マシンコード | モニタック | コメント |
|------|--------|-----------------|---------------------------------|
| D11B | 2A08E0 | LD HL, (0E008H) | ; ** E008H = ADR ** |
| D11E | 112800 | LD DE, 0028H | ; DE ← 40 |
| D121 | B7 | OR A | ; Cy ← 0 |
| D122 | ED52 | SBC HL, DE | ; HL ← HL - 40 |
| D124 | 44 | LD B, H | |
| D125 | 4D | LD C, L | ; BC ← HL |
| D126 | ED78 | IN A, (C) | ; A ← I/O (BC) |
| D128 | FE20 | CP 20H | ; スタック・コート カ? |
| D12A | 28 | JR Z, GOUP2 | ; スタック・コート カ? |
| D12C | B7 | OR A | ; Cy ← 0 ... スタック・コート カ リセット スル |
| D12D | C9 | RET | |

■ GOUP1 (デバウチングでUIニスルサブルーチン) ■ (図4-19②)

| アドレス | マシンコード | モニタック | コメント |
|------|--------|-----------------|-----------------------------------|
| D12E | 2A08E0 | LD HL, (0E008H) | |
| D131 | 112800 | LD DE, 0028H | |
| D134 | B7 | OR A | |
| D135 | ED52 | SBC HL, DE | ; コマンド・ハ、GOUP ト オナシ |
| D137 | 220AE0 | LD (0E00AH), HL | ; GOUP2 ** E00AH = NADR ** |
| D13A | 3E08 | LD A, 08H | |
| D13C | 320CE0 | LD (0E00CH), A | ; MOVE ニ「UI」ノ コート 8 ヲ イレ |
| D13F | 3A0DE0 | LD A, (0E00DH) | ; ** E00DH = TS ** |
| D142 | FE00 | CP 00H | ; トロノ カ? |
| D144 | 20 | JR NZ, サーク・UI | |
| D146 | 3A01E0 | LD A, (0E001H) | ; トロノ・UI ノ ショリ ** E001H = TRY ** |
| D149 | 3D | DEC A | |
| D14A | 3201E0 | LD (0E001H), A | ; TRY ノ アライ ← TRY ノ アライ - 1 |
| D14D | 18 | JR WI・ショリ オフリ | |
| D14F | 3A03E0 | LD A, (0E003H) | ; サーク・UI ノ ショリ ** E003H = SKY ** |
| D152 | 3D | DEC A | |
| D153 | 3203E0 | LD (0E003H), A | ; SKY ノ アライ ← SKY ノ アライ - 1 |
| D156 | 37 | SCF | ; UI・ショリ オフリ ... スタック・コート リセット スル |
| D157 | C9 | RET | |

■ GODN (ハンダマン ツキ ティ シタ ニ スス ヲ フアル-チン) ■ (図 4 - 2 0 ①)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|--------------|---|
| D158 | 2A08E0 | LD HL, 0E08H | ; *E08H = ADDR * |
| D15B | 112800 | LD DE, 0028H | ; DE ← 40 |
| D15E | 19 | ADD HL, DE | ; HL ← HL + 40 |
| D15F | 44 | LD B, H | |
| D160 | 4D | LD C, L | ; BC ← HL |
| D161 | ED78 | IN A, (C) | ; A ← I/O (BC) |
| D163 | FE20 | CP 20H | ; ステータス・コード カ ? |
| D165 | 28 | JR Z, GODN2 | ; スミル ナラ、GODN2 へ |
| D167 | B7 | OR A | ; Cy ← 0 ... ステータス・コード、Cy フラグ ヲ リセット スル |
| D168 | C9 | RET | |

■ GODN1 (ムシヨウケン ニ シタ ニ スス ヲ フアル-チン) ■ (図 4 - 2 0 ②)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|-----------------|-----------------------------------|
| D169 | 2A08E0 | LD HL, (0E08H) | |
| D16C | 112800 | LD DE, 0028H | |
| D16F | 19 | ADD HL, DE | ; コマンドハ、GODN ト オナシ |
| D170 | 220AE0 | LD (0E0AH), HL | ; GODN2 *E0AH = NADR * |
| D173 | 3E02 | LD A, 02H | |
| D175 | 320CE0 | LD (0E0CH), A | ; MOVE ニ 「シタ」 ノ コード 2 ヲ イレル |
| D178 | 3A0DE0 | LD A, (0E0DH) | ; *E0DH = TS * |
| D17B | FE00 | CP 00H | ; トロフ カ ? |
| D17D | 28 | JR NZ, サーク・シタ へ | |
| D17F | 3A01E0 | LD A, (0E01H) | ; トロフ・シタ ノ ショリ *E01H = TRY * |
| D182 | 3C | INC A | |
| D183 | 3201E0 | LD (0E01H), A | ; TRY ノ アライ ← TRY ノ アライ + 1 |
| D186 | 18 | JR シタ・ショリ オフリ へ | |
| D188 | 3A03E0 | LD A, (0E03H) | ; サーク・シタ ノ ショリ *E03H = SKY * |
| D18B | 3C | INC A | |
| D18C | 3203E0 | LD (0E03H), A | ; SKY ノ アライ ← SKY ノ アライ + 1 |
| D18F | 37 | SCF | ; シタ・ショリ オフリ ... Cy フラグ ヲ リセット スル |
| D190 | C9 | RET | |

■ GOLT (ハンダマン ツキ ティ ヒタリ ニ スス ヲ フアル-チン) ■ (図 4 - 2 1 ①)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|----------------|---|
| D191 | 2A08E0 | LD HL, (0E08H) | ; *E08H = ADDR * |
| D194 | 2B | DEC HL | ; HL ← HL - 1 |
| D195 | 44 | LD B, H | |
| D196 | 4D | LD C, L | ; BC ← HL |
| D197 | ED78 | IN A, (C) | ; A ← I/O (BC) |
| D199 | FE20 | CP 20H | ; ステータス・コード カ ? |
| D19B | 28 | JR Z, GOLT2 | ; スミル ナラ、GOLT2 へ |
| D19D | B7 | OR A | ; Cy ← 0 ... ステータス・コード、Cy フラグ ヲ リセット スル |
| D19E | C9 | RET | |

■■■ GOLT1 (ムシヨウケン ニ ヒタリ ニ スム サブルーチン) ■■■

(図4-21②)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|------------------|-----------------------------------|
| D19F | 2A08E0 | LD HL, (0E008H) | |
| D1A2 | 2B | DEC HL | ; コマテハ、GOLT ト オナシ |
| D1A3 | 220AE0 | LD (0E00AH), HL | ; GOLT2 ※※ E00AH = NADR ※※ |
| D1A6 | 3E04 | LD A, 04H | |
| D1A8 | 320CE0 | LD (0E00CH), A | ; MOVE ニ 「ヒタリ」 ノ コート 4 ヲ イレル |
| D1AB | 3A0DE0 | LD A, (0E00DH) | ; ※※ E00DH = TS ※※ |
| D1AE | FE00 | CP 00H | ; トロン カ ? |
| D1B0 | 20__ | JR NZ, サーク・ヒタリ ヲ | |
| D1B2 | 3A00E0 | LD A, (0E000H) | ; トロン・ヒタリ ノ ショリ ※※ E000H = TRX ※※ |
| D1B5 | 3D | DEC A | |
| D1B6 | 3200E0 | LD (0E000H), A | ; TRX ノ アライ <— TRX ノ アライ - 1 |
| D1B9 | 18__ | JR ヒタリ・ショリ オフリ ヲ | |
| D1BB | 3A02E0 | LD A, (0E002H) | ; サーク・ヒタリ ノ ショリ ※※ E002H = SKX ※※ |
| D1BE | 3D | DEC A | |
| D1BF | 3202E0 | LD (0E002H), A | ; SKX ノ アライ <— SKX ノ アライ - 1 |
| D1C2 | 37 | SCF | ; ヒタリ・ショリ オフリ .. Cフフラク ヲ セット スル |
| D1C3 | C9 | RET | |

■■■ GORT (ハンダダン ツキ テハ ミキ ニ スム サブルーチン) ■■■

(図4-22①)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|-----------------|---------------------------------------|
| D1C4 | 2A08E0 | LD HL, (0E008H) | ; ※※ E008H = ADR ※※ |
| D1C7 | 23 | INC HL | ; HL <— HL + 1 |
| D1C8 | 44 | LD B, H | |
| D1C9 | 4D | LD C, L | ; BC <— HL |
| D1CA | ED78 | IN A, (C) | ; A <— I/O (BC) |
| D1CC | FE20 | CP 20H | ; スハ・ス・コート カ ? |
| D1CE | 28__ | JR Z, GORT2 | ; スズメル ナラ、GORT2 ヲ |
| D1D0 | B7 | OR A | ; Cy <— 0 ... スズメタイアラ、Cフフラク ヲ リセット スル |
| D1D1 | C9 | RET | |

■■■ GORT1 (ムシヨウケン ニ ミキ ニ スム サブルーチン) ■■■

(図4-22②)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|-----------------|----------------------------------|
| D1D2 | 2A08E0 | LD HL, (0E008H) | |
| D1D5 | 23 | INC HL | ; コマテハ、GORT ト オナシ |
| D1D6 | 220AE0 | LD (0E00AH), HL | ; GORT2 ※※ E00AH = NADR ※※ |
| D1D9 | 3E06 | LD A, 06H | |
| D1DB | 320CE0 | LD (0E00CH), A | ; MOVE ニ 「ミキ」 ノ コート 6 ヲ イレル |
| D1DE | 3A0DE0 | LD A, (0E00DH) | ; ※※ E00DH = TS ※※ |
| D1E1 | FE00 | CP 00H | ; トロン カ ? |
| D1E3 | 20__ | JR NZ, サーク・ミキ ヲ | |
| D1E5 | 3A00E0 | LD A, (0E000H) | ; トロン・ミキ ノ ショリ ※※ E000H = TRX ※※ |
| D1E8 | 3C | INC A | |
| D1E9 | 3200E0 | LD (0E000H), A | ; TRX ノ アライ <— TRX ノ アライ + 1 |
| D1EC | 18__ | JR ミキ・ショリ オフリ ヲ | |
| D1EE | 3A02E0 | LD A, (0E002H) | ; サーク・ミキ ノ ショリ ※※ E002H = SKX ※※ |
| D1F1 | 3C | INC A | |
| D1F2 | 3202E0 | LD (0E002H), A | ; SKX ノ アライ <— SKX ノ アライ + 1 |
| D1F5 | 37 | SCF | ; ミキ・ショリ オフリ .. Cフフラク ヲ セット スル |
| D1F6 | C9 | RET | |

4-18／表示ルーチンCHRPRの作成

続けて、マシン語ルーチン後半部の作成にかかりましょう。まず、移動ルーチンからのジャンプ先に指定しておいた表示ルーチン CHRPR を配置いたします。

4-12節で設計した方針で、CHRPR ルーチンは、リスト (図4-23) のように書き下してみました。

1つ注意すべき点を述べます。それは、プリント・ルーチン (PRINT と名づける) へのデータの引き渡し方です。ここでは、4-14節で触れたように、レジスタを介してデータを渡す方法を採用してみました。すなわち、PRINTサブルーチンは次の仕様といたします。

《PRINTサブルーチンの仕様》

入力条件 : Eレジスタ = 表示したいキャラクターのASCIIコード

Dレジスタ = キャラクター識別コード (00H=軌跡キャラクター,
01H=バイクキャラクター, 02H=衝突キャラクター)

BCレジスタペア = 表示位置のVRAMアドレス

機能 : DレジスタとTSフラグ (トロン・サーク識別フラグ) を参照して適切なアトリビュートを決定し、Eレジスタのコードを、BCレジスタペアのVRAMアドレスに出力する。

このことを念頭に置いて、リストを眺めると、何をしているかがわかると思います。

また、PRINTルーチンへジャンプする所がありますが、それは、サブルーチン内では、

```
J P PRINT = CALL PRINT
RET
```

という同値性が成立することによっています。

図 4 - 2 3

CHRPR (キャラクター ヒョウシバ サブルーチン)

| アドレス | マシンコード | ニーモニック | コメント |
|------|----------|---------------------|---|
| D1F7 | CD | CALL CHRDET | ; キャラクター ノ センテイ ルーチン ヲ ヨビダス |
| D1FA | ED4B08E0 | LD BC, (0E008H) | ; *OK E008H = ADDR *OK |
| D1FE | 1600 | LD D, 00H | ; *セキ ノ キャラクター ヲ シテイ |
| D200 | 3A0EE0 | LD A, (0E00EH) | ; *OK E00EH = ORBIT *OK |
| D203 | 5F | LD E, A | ; E <— *セキ ノ ASCII コード |
| D204 | CD | CALL PRINT | ; プリント ルーチン ヲ ヨビダス |
| D207 | 3A07E0 | LD A, (0E007H) | ; *OK E007H = FLAG *OK |
| D20A | FE00 | CP 00H | ; ショウトツ カ ? |
| D20C | 20 | JR NZ, ショウトツ・キャラクター | ショリ ^ |
| D20E | ED4B0AE0 | LD BC, (0E00AH) | ; ハイク・キャラクター ショリ *OK E00AH = NADR *OK |
| D212 | 1601 | LD D, 01H | ; ハイク ノ キャラクター ヲ シテイ |
| D214 | 3A0FE0 | LD A, (0E00FH) | ; *OK E00FH = BIKE *OK |
| D217 | 5F | LD E, A | ; E <— ハイク ノ ASCII コード |
| D218 | C3 | JP PRINT | ; プリント ルーチン ^ |
| D21B | ED4B0AE0 | LD BC, (0E00AH) | ; ショウトツ・キャラクター ショリ *OK E00AH = NADR *OK |
| D21F | 1602 | LD D, 02H | ; ショウトツ ノ キャラクター ヲ シテイ |
| D221 | 1EE8 | LD E, 0E8H | ; E <— X ノ ASCII コード (E8H) |
| D223 | C3 | JP PRINT | ; プリント ルーチン ^ |

4-19/キャラクター選定サブルーチンCHRDETの作成

次に CHRPR から呼び出される1つ目の下位サブルーチン CHRDET を書き下しましょう。4-12節で設計した仕様に従って作成します。

続いて、下位サブルーチン CHRUP, CHRDN, CHRLT, CHRRT を配置いたします。

リスト (図4-24から図4-27) のようになります。

図4-24

CHRDET (キャラクター ケッタイ ルーチン)

| アドレス | マシンコード | ニーモニック | コメント |
|------|---------|-----------------------|-------------------------------------|
| D226 | 3A0CE0 | LD A, (0E00CH) | ; *E00CH = MOVE *K |
| D229 | FE02 | CP 02H | ; イトウ ホウゴウヲ シラゲル |
| D22B | CA_____ | JP Z, シラ・キャラクター ヲ | ; ホウゴウ カ 2 ノ トキ |
| D22E | FE04 | CP 04H | |
| D230 | CA_____ | JP Z, ヒタバリ・キャラクター ヲ | ; ホウゴウ カ 4 ノ トキ |
| D233 | FE06 | CP 06H | |
| D235 | CA_____ | JP Z, ミキ・キャラクター ヲ | ; ホウゴウ カ 6 ノ トキ |
| D238 | 3A0DE0 | LD A, (0E00DH) | ; ウィ・キャラクター (CHRUP) *E00DH = TS *K |
| D23B | FE00 | CP 00H | ; トロン カ ? |
| D23D | 20_____ | JR NZ, サーク・ウィ・キセキ ヲ | |
| D23F | 3A04E0 | LD A, (0E004H) | ; トロン・ウィ・キセキ ノ ショリ *E004H = TRMV *K |
| D242 | 18_____ | JR 「ヒタバリ カラ ウィ ?」 ヲ | |
| D244 | 3A05E0 | LD A, (0E005H) | ; サーク・ウィ・キセキ ノ ショリ *E005H = SKMV *K |
| D247 | FE04 | CP 04H | ; 「ヒタバリ カラ ウィ ?」 |
| D249 | 20_____ | JR NZ, 「ミキ カラ ウィ ?」 ヲ | |
| D24B | 3E99 | LD A, 99H | ; A ← 9 ノ コード |
| D24D | 18_____ | JR ウィ・キセキ ヲ | |
| D24F | FE06 | CP 06H | ; 「ミキ カラ ウィ ?」 |
| D251 | 20_____ | JR NZ, 「ウィ ノ ママ」 ヲ | |
| D253 | 3E98 | LD A, 98H | ; A ← 8 ノ コード |
| D255 | 18_____ | JR ウィ・キセキ ヲ | |
| D257 | 3E91 | LD A, 91H | ; 「ウィ ノ ママ」 .. A ← 1 ノ コード |
| D259 | 320EE0 | LD (0E00EH), A | ; ウィ・キセキ ノ ショリ *E00EH = ORBIT *K |
| D25C | 3A0DE0 | LD A, (0E00DH) | ; *E00DH = TS *K |
| D25F | FE00 | CP 00H | ; トロン カ ? |
| D261 | 20_____ | JR NZ, サーク・ウィ・ハイク ヲ | |
| D263 | 3E64 | LD A, 64H | ; トロン・ウィ・ハイク .. 64H = 100 |
| D265 | 320FE0 | LD (0E00FH), A | ; BIKE = 100ヲ カクノ スル |
| D268 | 3E08 | LD A, 08H | ; *K 8 = ウィ ノ コード *K |
| D26A | 3204E0 | LD (0E004H), A | ; TRMV = 8ヲ カクノ スル |
| D26D | C9 | RET | |
| D26E | 3EC8 | LD A, 0C8H | ; サーク・ウィ・ハイク .. C8H = 200 |
| D270 | 320FE0 | LD (0E00FH), A | ; BIKE = 200ヲ カクノ スル |
| D273 | 3E08 | LD A, 08H | ; *K 8 = ウィ ノ コード *K |
| D275 | 3205E0 | LD (0E005H), A | ; SKMV = 8ヲ カクノ スル |
| D278 | C9 | RET | |

| アドレス | マシンコード | メモリー | コメント |
|------|--------|------------------------|---------------------------------------|
| D279 | 3A0DE0 | LD A, (0E00DH) | ; シタ・キトラクター (CHRDN) ※※ E00DH = TS ※※ |
| D27C | FE00 | CP 00H | ; トロン カ ? |
| D27E | 20__ | JR NZ, サーク・シタ・キセキ ヲ | |
| D280 | 3A04E0 | LD A, (0E004H) | ; トロン・シタ・キセキ ノ ショリ ※※ E004H = TRMV ※※ |
| D283 | 18__ | JR 「ヒタリ カラ シタ ?」 ヲ | |
| D285 | 3A05E0 | LD A, (0E005H) | ; サーク・シタ・キセキ ノ ショリ ※※ E005H = SKMV ※※ |
| D288 | FE04 | CP 04H | ; 「ヒタリ カラ シタ ?」 |
| D28A | 20__ | JR NZ, 「ミキ」 カラ シタ ?」 ヲ | |
| D28C | 3E9A | LD A, 9AH | ; A ← 9 / コート |
| D28E | 18__ | JR シタ・キセキ ヲ | |
| D290 | FE06 | CP 06H | ; 「ミキ」 カラ シタ ?」 |
| D292 | 20__ | JR NZ, 「シタ ノ ママ」 ヲ | |
| D294 | 3E97 | LD A, 97H | ; A ← 7 / コート |
| D296 | 18__ | JR シタ・キセキ ヲ | |
| D298 | 3E91 | LD A, 91H | ; 「シタ ノ ママ」 .. A ← 1 / コート |
| D29A | 320EE0 | LD (0E00EH), A | ; シタ・キセキ ノ ショリ ※※ E00EH = ORBIT ※※ |
| D29D | 3A0DE0 | LD A, (0E00DH) | ; ※※ E00DH = TS ※※ |
| D2A0 | FE00 | CP 00H | ; トロン カ ? |
| D2A2 | 20__ | JR NZ, サーク・シタ・ハイク ヲ | |
| D2A4 | 3E6E | LD A, 6EH | ; トロン・シタ・ハイク .. 6EH = 110 |
| D2A6 | 320FE0 | LD (0E00FH), A | ; BIKE ニ 110 ヲ カクノウ スル |
| D2A9 | 3E02 | LD A, 02H | ; ※※ 2 = シタ ノ コート ※※ |
| D2AB | 3204E0 | LD (0E004H), A | ; TRMV ニ 2 ヲ カクノウ スル |
| D2AE | C9 | RET | |
| D2AF | 3ED2 | LD A, 0D2H | ; サーク・シタ・ハイク .. D2H = 210 |
| D2B1 | 320FE0 | LD (0E00FH), A | ; BIKE ニ 210 ヲ カクノウ スル |
| D2B4 | 3E02 | LD A, 02H | ; ※※ 2 = シタ ノ コート ※※ |
| D2B6 | 3205E0 | LD (0E005H), A | ; SKMV ニ 2 ヲ カクノウ スル |
| D2B9 | C9 | RET | |

図4-25 (CHRDN)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|------------------------|---------------------------------------|
| D2BA | 3A0DE0 | LD A, (0E00DH) | ;ヒタリ・キャラクター (CHRLT) ※※ E00DH = TS ※※ |
| D2BD | FE00 | CP 00H | ;トロン カ ? |
| D2BF | 20__ | JR NZ, ワーク・ヒタリ・キセキ ^ | |
| D2C1 | 3A04E0 | LD A, (0E004H) | ;トロン・ヒタリ・キセキ / ショリ ※※ E004H = TRMV ※※ |
| D2C4 | 18__ | JR 「UI カラ ヒタリ ?」 ^ | |
| D2C6 | 3A05E0 | LD A, (0E005H) | ;ワーク・ヒタリ・キセキ / ショリ ※※ E005H = SKMV ※※ |
| D2C9 | FE08 | CP 08H | ;「UI カラ ヒタリ ?」 |
| D2CB | 20__ | JR NZ, 「シタ カラ ヒタリ ?」 ^ | |
| D2CD | 3E97 | LD A, 97H | ;A <— 7 / コート |
| D2CF | 18__ | JR ヒタリ・キセキ ^ | |
| D2D1 | FE02 | CP 02H | ;「シタ カラ ヒタリ ?」 |
| D2D3 | 20__ | JR NZ, 「ヒタリ / ママ」 ^ | |
| D2D5 | 3E98 | LD A, 98H | ;A <— 8 / コート |
| D2D7 | 18__ | JR ヒタリ・キセキ ^ | |
| D2D9 | 3E90 | LD A, 90H | ;「ヒタリ / ママ」 .. A <— 0 / コート |
| D2DB | 320EE0 | LD (0E00EH), A | ;ヒタリ・キセキ / ショリ ※※ E00EH = ORBIT ※※ |
| D2DE | 3A0DE0 | LD A, (0E00DH) | ; ※※ E00DH = TS ※※ |
| D2E1 | FE00 | CP 00H | ;トロン カ ? |
| D2E3 | 20__ | JR NZ, ワーク・ヒタリ・ハイク ^ | |
| D2E5 | 3E78 | LD A, 78H | ;トロン・ヒタリ・ハイク .. 78H = 120 |
| D2E7 | 320FE0 | LD (0E00FH), A | ;BIKE ニ 120 ヲ カクノウ スル |
| D2EA | 3E04 | LD A, 04H | ; ※※ 4 = ヒタリ / コート ※※ |
| D2EC | 3204E0 | LD (0E004H), A | ;TRMV ニ 4 ヲ カクノウ スル |
| D2EF | C9 | RET | |
| D2F0 | 3EDC | LD A, 0DCH | ;ワーク・ヒタリ・ハイク .. DCH = 220 |
| D2F2 | 320FE0 | LD (0E00FH), A | ;BIKE ニ 220 ヲ カクノウ スル |
| D2F5 | 3E04 | LD A, 04H | ; ※※ 4 = ヒタリ / コート ※※ |
| D2F7 | 3205E0 | LD (0E005H), A | ;SKMV ニ 4 ヲ カクノウ スル |
| D2FA | C9 | RET | |

図4-26 (CHRLT)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|------------------------|--|
| D2FB | 3A0DE0 | LD A, (0E00DH) | ; ミキ°・キヤラクダ° (CHRRT) ** E00DH = TS ** |
| D2FE | FE00 | CP 00H | ; トロン カ ? |
| D300 | 20__ | JR NZ, サーク・ミキ°・キセキ ° | |
| D302 | 3A04E0 | LD A, (0E004H) | ; トロン・ミキ°・キセキ / ショリ ** E004H = TRMV ** |
| D305 | 18__ | JR 「ウエ カラ ミキ° ?」 ° | |
| D307 | 3A05E0 | LD A, (0E005H) | ; サーク・ミキ°・キセキ / ショリ ** E005H = SKMV ** |
| D30A | FE08 | CP 08H | ; 「ウエ カラ ミキ° ?」 |
| D30C | 20__ | JR NZ, 「シタ カラ ミキ° ?」 ° | |
| D30E | 3E9A | LD A, 9AH | ; A <— r / コート° |
| D310 | 18__ | JR ミキ°・キセキ ° | |
| D312 | FE02 | CP 02H | ; 「シタ カラ ミキ° ?」 |
| D314 | 20__ | JR NZ, 「ミキ° / ママ」 ° | |
| D316 | 3E99 | LD A, 99H | ; A <— r / コート° |
| D318 | 18__ | JR ミキ°・キセキ ° | |
| D31A | 3E98 | LD A, 90H | ; 「ミキ° / ママ」 ° A <— r / コート° |
| D31C | 320EE0 | LD (0E00EH), A | ; ミキ°・キセキ / ショリ ** E00EH = ORBIT ** |
| D31F | 3A0DE0 | LD A, (0E00DH) | ; ** E00DH = TS ** |
| D322 | FE00 | CP 00H | ; トロン カ ? |
| D324 | 20__ | JR NZ, トロン・ミキ°・ハ°イク ° | |
| D326 | 3E82 | LD A, 82H | ; トロン・ミキ°・ハ°イク ° 82H = 130 |
| D328 | 320FE0 | LD (0E00FH), A | ; BIKE に 130 ラ カクノウ スル |
| D32B | 3E06 | LD A, 06H | ; ** 6 = ミキ° / コート° ** |
| D32D | 3204E0 | LD (0E004H), A | ; TRMV に 6 ラ カクノウ スル |
| D330 | C9 | RET | |
| D331 | 3EE6 | LD A, 0E6H | ; サーク・ミキ°・ハ°イク ° E6H = 230 |
| D333 | 320FE0 | LD (0E00FH), A | ; BIKE に 230 ラ カクノウ スル |
| D336 | 3E06 | LD A, 06H | ; ** 6 = ミキ° / コート° ** |
| D338 | 3205E0 | LD (0E005H), A | ; SKMV に 6 ラ カクノウ スル |
| D33B | C9 | RET | |

図4-27 (CHRRT)

4-20/プリントルーチンPRINTの作成

表示ルーチンの最後に、下位サブルーチン PRINT を配置します。4-18節で決めた仕様に従って書き下すと、リスト (図4-28) のようになりました。

この段階で、今まで未定だった絶対アドレス、相対アドレスはすべて決定されているはずですから、空欄を記入して埋めておいて下さい。

図4-28

PRINT (プリント サブルーチン)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|--------------------|---|
| D33C | 7A | LD A, D | ;A ←— キャラクタ— シキヘツ コード |
| D33D | FE01 | CP 01H | ;キャラクタ— の バイカ ? |
| D33F | 20 | JR NZ, キセキ・アトリビュート | ↑ |
| D341 | 3E27 | LD A, 27H | ;バイカ・アトリビュート .. COLOR 7, CGEN 1 |
| D343 | 18 | JR アトリビュート・シュツリョク | ↑ |
| D345 | 3A0DE0 | LD A, (0E00DH) | ;キセキ・アトリビュート .. ** E00DH = TS ** |
| D348 | FE00 | CP 00H | ;トロン カ ? |
| D34A | 20 | JR NZ, ワーク・アトリビュート | ↑ |
| D34C | 7A | LD A, D | ;トロン・アトリビュート .. A ←— キャラクタ— シキヘツ コード |
| D34D | FE00 | CP 00H | ;キセキ カ ? |
| D34F | 20 | JR NZ, トロン・ショウトツ | ↑ |
| D351 | 3E26 | LD A, 26H | ;トロン・キセキ・アトリビュート .. COLOR 6, CGEN 1 |
| D353 | 18 | JR アトリビュート・シュツリョク | ↑ |
| D355 | 3E06 | LD A, 06H | ;トロン・ショウトツ .. COLOR 6, CGEN 0 |
| D357 | 18 | JR アトリビュート・シュツリョク | ↑ |
| D359 | 7A | LD A, D | ;ワーク・アトリビュート .. A ←— キャラクタ— シキヘツ コード |
| D35A | FE00 | CP 00H | ;キセキ カ ? |
| D35C | 20 | JR NZ, ワーク・ショウトツ | ↑ |
| D35E | 3E22 | LD A, 22H | ;ワーク・キセキ・アトリビュート .. COLOR 2, CGEN 1 |
| D360 | 18 | JR アトリビュート・シュツリョク | ↑ |
| D362 | 3E02 | LD A, 02H | ;ワーク・ショウトツ .. COLOR 2, CGEN 0 |
| D364 | CBA0 | RES 4, B | ;アトリビュート・シュツリョク .. BC = アトリビュート URAM アドレス |
| D366 | ED79 | OUT (C), A | ;I/O (BC) ←— A .. A = アトリビュート・コード |
| D368 | CBE0 | SET 4, B | ;キャラクタ—・シュツリョク .. BC = テキスト URAM アドレス |
| D36A | 7B | LD A, E | ;A ←— キャラクタ— / ASCII コード |
| D36B | ED79 | OUT (C), A | ;I/O (BC) ←— A |
| D36D | C9 | RET | |

4-21 / 画面反転サブルーチン(CREV)の配置

いよいよマシン語プログラムの作成も最後を迎えました。PRINTサブルーチンは、D36DH番地で終わりましたから、このすぐ後 D36EH 番地から、画面反転サブルーチンを格納いたします。これはすでに前章で完成したものを利用いたします。リロケータブルになっていましたから、そのままでもいいですね。

リスト図4-29

CREV (カメン ハンテン サブルーチン)

| アドレス | マシンコード | ニーモニック | コメント |
|------|--------|--------------|-------|
| D36E | 010020 | LD BC, 2000H | |
| D371 | ED78 | IN A, (C) | ; ルーフ |
| D373 | CBDF | SET 3, A | |
| D375 | ED79 | OUT (C), A | |
| D377 | 03 | INC BC | |
| D378 | 78 | LD A, B | |
| D379 | FE23 | CP 23H | |
| D37B | 38F4 | JR C, ルーフ ↑ | |
| D37D | 79 | LD A, C | |
| D37E | FEE8 | CP 0E8H | |
| D380 | 38EF | JR C, ルーフ ↑ | |
| D382 | C9 | RET | |

以上をもちまして、すべてのサブルーチンが完成いたしました。本当に御苦労様でした。

□コーヒープレイク

マシン語プログラムの作成過程がわかるように記述したつもりですが、雰囲気は伝わったでしょうか？

この程度の長さのプログラムでも、ハンド・アセンブルですと、少しウンザリするというのが実感だと思います。特に相対アドレスの計算は面倒ですね。

X1用のアセンブラがあったらなあ、とは誰もが考える所ですが、私の知る限りまだ、満足すべきアセンブラは発表されていないようです。

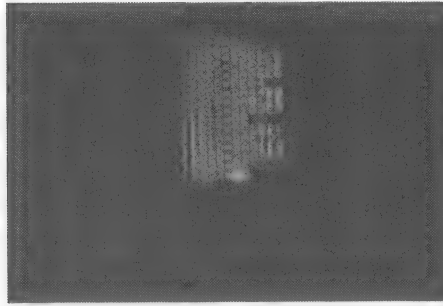
唯一、「I/O」誌の1983年3月号に dB SOFT から「ミニアセンブラ」が発表されております。このアセンブラはBASICのREM文でニーモニックを入力し、マシン語アセンブラサブルーチンをCALLすることでアセンブルする形のものですが、書式が少し違う点、大きなプログラムが組めない点、時々原因不明のアセンブル・エラーを生ずる点など、まだまだ満足できるものではありません。1ユーザーとして、オールマシン語版のエディタ・アセンブラを望む所です。

けれど… 不満を言っても今はこれしかないのだから！ と思い直して、私はトロン・ゲームのマシン語部分を「ミニアセンブラ」で作成してみたのです。こわごわだったのですが、幸いなことに何とか動いてくれて、ニーモニックはキッチンとマシンコードに変換されました。参考までに、私の作成した「ミニアセンブラ」用アセンブルリストを付録5に挙げておきましたので、御覧下さい。

4-22/チェツフ・サムの使い方

さて、プログラムが書き上がりましたので、いよいよ、モニターを起動して、メモリーに書き込んでゆきます。ニーモニック、コメントは無視して、アドレスを参照しながら、*M コマンドにより、マシンコードを入力して行って下さい。

図4-30



D000H番地から、D382H番地まで入力終了しましたか？

これが終わりましたら、取りあえずテープにセーブしておいて下さい。*S コマンドで、

*S D000 D382 D000 : TRON MAS

とでもすれば、D000H～D382H番地のメモリー内容がテープにセーブされます。

マシン語プログラムの場合、入力ミスエラーがあっても、システムはエラー表示をしてくれず、テスト実行でも「暴走」によりプログラム本体を壊すことがありますので、一応セーブしておく心掛けは、BASICプログラム以上に大切です。

さて、セーブが終了しましたら、入力ミスのチェックに移ります。これは、自分の作成したプログラムの場合は大変で、プログラムを小部分に分け、テストプログラムを通して実行確認していかなばなりません。本当は、開発の雰囲気伝えるのに、部分部分のテストプログラムを掲載して読者の皆様にも実験していただくとよいのですが、ページ数が莫大になりそうなので、ここでは省略します。

かわりに、すでに完成品がどこかにあるとして、それと比較してミスチェックをする方法をとしましょう。マイコン雑誌のマシン語プログラムでは大抵この方法をとっていますね。この時、必ず登場するものに、チェックサム (Check Sum) という表があります。次に掲げるのが、私たちが作成したトロンゲームのマシン語プログラム部分のチェックサムです。(図4-31①～図4-31④)

图4-31①

```

:D000 = 3E 01 32 0D E0 CD EC D0 : E7
:D008 = 3A 06 E0 FE 04 DA 44 D0 : 10
:D010 = 3A 01 E0 21 03 E0 BE DC : B9
:D018 = 1B D1 DA F7 D1 3A 01 E0 : A9
:D020 = 21 03 E0 BE C4 58 D1 DA : 89
:D028 = F7 D1 3A 00 E0 21 02 E0 : E5
:D030 = BE DC 91 D1 DA F7 D1 3A : D8
:D038 = 00 E0 21 02 E0 BE C4 C4 : 29
:D040 = D1 DA F7 D1 CD 1B D1 DA : 06
:D048 = F7 D1 CD 58 D1 DA F7 D1 : 60
:D050 = CD 91 D1 DA F7 D1 CD C4 : 62
:D058 = D1 DA F7 D1 3A 0C E0 FE : 97
:D060 = 08 20 05 CD 2E D1 18 15 : 26
:D068 = FE 02 20 05 CD 69 D1 18 : 44
:D070 = 0C FE 04 20 05 CD 9F D1 : 70
:D078 = 18 03 CD D2 D1 3E 01 32 : FC

```

```

33 A2 1A 4C B6 06 55 B1 : FD

```

```

:D080 = 07 E0 C3 F7 D1 3E 00 32 : E2
:D088 = 0D E0 CD EC D0 3A 04 E0 : 94
:D090 = 21 0C E0 86 FE 0A 20 03 : BE
:D098 = 21 04 E0 7E FE 08 20 10 : B9
:D0A0 = CD 1B D1 38 08 CD 2E D1 : C5
:D0A8 = 3E 01 32 07 E0 C3 F7 D1 : E3
:D0B0 = FE 02 20 10 CD 58 D1 38 : 5E
:D0B8 = 08 CD 69 D1 3E 01 32 07 : 87
:D0C0 = E0 C3 F7 D1 FE 04 20 10 : 9D
:D0C8 = CD 91 D1 38 08 CD 9F D1 : AC
:D0D0 = 3E 01 32 07 E0 C3 F7 D1 : E3
:D0D8 = FE 06 20 BC CD C4 D1 38 : 7A
:D0E0 = 08 CD D2 D1 3E 01 32 07 : F0
:D0E8 = E0 C3 F7 D1 3A 0D E0 FE : 90
:D0F0 = 00 20 0C 7E 32 0C E0 21 : E9
:D0F8 = 00 E0 56 23 5E 18 0C 3A : 15

```

```

38 A6 21 16 4B FD F1 50 : 9E

```

图4 - 3 1②

```

:D100 = 05 E0 32 0C E0 21 02 E0 : 06
:D108 = 56 23 5E 21 00 30 4A 16 : 88
:D110 = 00 06 28 19 10 FD 09 22 : 7F
:D118 = 08 E0 C9 2A 08 E0 11 28 : FC
:D120 = 00 B7 ED 52 44 4D ED 78 : EC
:D128 = FE 20 28 0B B7 C9 2A 08 : 03
:D130 = E0 11 28 00 B7 ED 52 22 : 31
:D138 = 0A E0 3E 08 32 0C E0 3A : 88
:D140 = 0D E0 FE 00 20 09 3A 01 : 4F
:D148 = E0 3D 32 01 E0 18 07 3A : 89
:D150 = 03 E0 3D 32 03 E0 37 C9 : 35
:D158 = 2A 08 E0 11 28 00 19 44 : A8
:D160 = 4D ED 78 FE 20 28 09 B7 : B8
:D168 = C9 2A 08 E0 11 28 00 19 : 2D
:D170 = 22 0A E0 3E 02 32 0C E0 : 6A
:D178 = 3A 0D E0 FE 00 20 09 3A : 88

```

D7 E4 89 33 3A E0 5E 4E : 3D

```

:D180 = 01 E0 3C 32 01 E0 18 07 : 4F
:D188 = 3A 03 E0 3C 32 03 E0 37 : A5
:D190 = C9 2A 08 E0 2B 44 4D ED : 84
:D198 = 78 FE 20 28 06 B7 C9 2A : 6E
:D1A0 = 08 E0 2B 22 0A E0 3E 04 : 61
:D1A8 = 32 0C E0 3A 0D E0 FE 00 : 43
:D1B0 = 20 09 3A 00 E0 3D 32 00 : B2
:D1B8 = E0 18 07 3A 02 E0 3D 32 : 8A
:D1C0 = 02 E0 37 C9 2A 08 E0 23 : 17
:D1C8 = 44 4D ED 78 FE 20 28 06 : 42
:D1D0 = B7 C9 2A 08 E0 23 22 0A : E1
:D1D8 = E0 3E 06 32 0C E0 3A 0D : 89
:D1E0 = E0 FE 00 20 09 3A 00 E0 : 21
:D1E8 = 3C 32 00 E0 18 07 3A 02 : A9
:D1F0 = E0 3C 32 02 E0 37 C9 CD : FD
:D1F8 = 26 D2 ED 4B 08 E0 16 00 : 2E

```

B5 8A 03 D4 7A 3E 36 7A : 7E

图 4 - 3 1 ③

```

:D200 = 3A 0E E0 5F CD 3C D3 3A : 9D
:D208 = 07 E0 FE 00 20 0D ED 4B : 4A
:D210 = 0A E0 16 01 3A 0F E0 5F : 89
:D218 = C3 3C D3 ED 4B 0A E0 16 : 0A
:D220 = 02 1E E8 C3 3C D3 3A 0C : 20
:D228 = E0 FE 02 CA 79 D2 FE 04 : F7
:D230 = CA BA D2 FE 06 CA FB D2 : F1
:D238 = 3A 0D E0 FE 00 20 05 3A : 84
:D240 = 04 E0 18 03 3A 05 E0 FE : 1C
:D248 = 04 20 04 3E 99 18 0A FE : 1F
:D250 = 06 20 04 3E 98 18 02 3E : 58
:D258 = 91 32 0E E0 3A 0D E0 FE : D6
:D260 = 00 20 0B 3E 64 32 0F E0 : EE
:D268 = 3E 08 32 04 E0 C9 3E C8 : 2B
:D270 = 32 0F E0 3E 08 32 05 E0 : 7E
:D278 = C9 3A 0D E0 FE 00 20 05 : 13

```

```

CC B0 BB 95 1C 60 F6 DB : 19

```

```

:D280 = 3A 04 E0 18 03 3A 05 E0 : 58
:D288 = FE 04 20 04 3E 9A 18 0A : 20
:D290 = FE 06 20 04 3E 97 18 02 : 17
:D298 = 3E 91 32 0E E0 3A 0D E0 : 16
:D2A0 = FE 00 20 0B 3E 6E 32 0F : 16
:D2A8 = E0 3E 02 32 04 E0 C9 3E : 3D
:D2B0 = D2 32 0F E0 3E 02 32 05 : 6A
:D2B8 = E0 C9 3A 0D E0 FE 00 20 : EE
:D2C0 = 05 3A 04 E0 18 03 3A 05 : 7D
:D2C8 = E0 FE 08 20 04 3E 97 18 : F7
:D2D0 = 0A FE 02 20 04 3E 98 18 : 1C
:D2D8 = 02 3E 90 32 0E E0 3A 0D : 37
:D2E0 = E0 FE 00 20 0B 3E 78 32 : F1
:D2E8 = 0F E0 3E 04 32 04 E0 C9 : 10
:D2F0 = 3E DC 32 0F E0 3E 04 32 : AF
:D2F8 = 05 E0 C9 3A 0D E0 FE 00 : D3

```

```

27 E6 94 17 17 B2 6C AD : 9A

```


☒ 4 - 3 1 ④

```

:D300 = 20 05 3A 04 E0 18 03 3A : 98
:D308 = 05 E0 FE 08 20 04 3E 9A : E7
:D310 = 18 0A FE 02 20 04 3E 99 : 1D
:D318 = 18 02 3E 90 32 0E E0 3A : 42
:D320 = 0D E0 FE 00 20 0B 3E 82 : D6
:D328 = 32 0F E0 3E 06 32 04 E0 : 7B
:D330 = C9 3E E6 32 0F E0 3E 06 : 52
:D338 = 32 05 E0 C9 7A FE 01 20 : 79
:D340 = 04 3E 27 18 1F 3A 0D E0 : C7
:D348 = FE 00 20 0D 7A FE 00 20 : C3
:D350 = 04 3E 26 18 0F 3E 06 18 : EB
:D358 = 0B 7A FE 00 20 04 3E 22 : 07
:D360 = 18 02 3E 02 CB A0 ED 79 : 2B
:D368 = CB E0 7B ED 79 C9 01 00 : 56
:D370 = 20 ED 78 CB DF ED 79 03 : 98
:D378 = 78 FE 23 38 F4 79 FE E8 : 24

```

```

1B E6 D7 06 E0 92 96 CD : B3

```

```

:D380 = 38 EF C9 00 00 00 00 00 : F0
:D388 = 00 00 00 00 00 00 00 00 : 00
:D390 = 00 00 00 00 00 00 00 00 : 00
:D398 = 00 00 00 00 00 00 00 00 : 00
:D3A0 = 00 00 00 00 00 00 00 00 : 00
:D3A8 = 00 00 00 00 00 00 00 00 : 00
:D3B0 = 00 00 00 00 00 00 00 00 : 00
:D3B8 = 00 00 00 00 00 00 00 00 : 00
:D3C0 = 00 00 00 00 00 00 00 00 : 00
:D3C8 = 00 00 00 00 00 00 00 00 : 00
:D3D0 = 00 00 00 00 00 00 00 00 : 00
:D3D8 = 00 00 00 00 00 00 00 00 : 00
:D3E0 = 00 00 00 00 00 00 00 00 : 00
:D3E8 = 00 00 00 00 00 00 00 00 : 00
:D3F0 = 00 00 00 00 00 00 00 00 : 00
:D3F8 = 00 00 00 00 00 00 00 00 : 00

```

```

38 EF C9 00 00 00 00 00 : F0

```

チェックサムの形式はいろいろあって、雑誌によっても異なりますが、本書で採用したのは、BASICのモニターのダンプ形式を真似た体裁です。欄外の右にあるのは、横一列（8バイト分）の和の16進下2桁を表示しており、下にあるのは、縦一列（16バイト分）の和の16進下2桁を表示しています。右下隅のは、トータルサムといって、 $8 \times 16 = 128$ バイト分の総和の下2桁を意味しています。

この表を活用するためには、読者の入力したマシン語部分について、チェックサムを表示（できれば同一形式で！）しなくてはなりません。

付録3に、「チェックサムプログラム」を載せておきましたので、入力してみるとよいと思います。自前のものですが、なかなか便利ですよ！

さて、「チェックサムプログラム」が完成しましたら、先程テープにセーブしておいたトロングゲームのマシン語部分をロードいたします。念のため、CLEAR &HD000を実行しておいた方が安全です。しかる後に、LOADM ないしは、モニター起動後の *L コマンド によりマシン語をロードして下さい。

さあ準備完了です。チェックサムプログラムをRUNして下さい。すべての注意は画面に表示されますから、指示通りにキーを操作して下さい。開始アドレス=D000，終了アドレス=D382 と指定し、リターン・キーを押すと、最初の 128バイト分のチェックサム画面が現われます。

図4-32

```

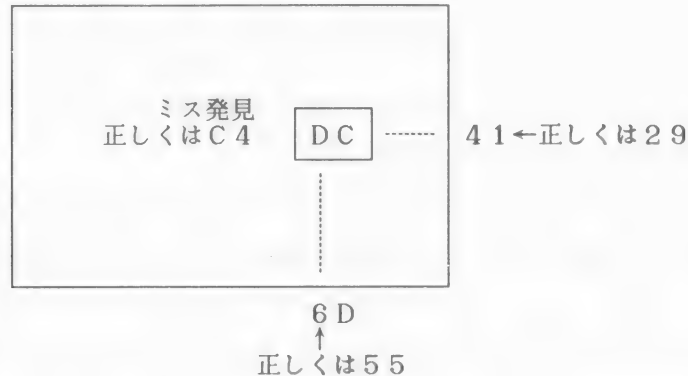
:D000 = 3E 01 32 0D E0 CD EC D0 : E7
:D008 = 3A 06 E0 FE 04 DA 44 D0 : 10
:D010 = 3A 01 E0 21 03 E0 BE DC : B9
:D018 = 1B D1 DA F7 D1 3A 01 E0 : A9
:D020 = 21 03 E0 BE C4 58 D1 DA : 89
:D028 = F7 D1 3A 00 E0 21 02 E0 : E5
:D030 = BE DC 91 D1 DA F7 D1 3A : D8
:D038 = 00 E0 21 02 E0 BE DC C4 : 41
:D040 = D1 DA F7 D1 CD 1B D1 DA : 06
:D048 = F7 D1 CD 58 D1 DA F7 D1 : 60
:D050 = CD 91 D1 DA F7 D1 CD C4 : 62
:D058 = D1 DA F7 D1 3A 0C E0 FE : 97
:D060 = 08 20 05 CD 2E D1 18 15 : 26
:D068 = FE 02 20 05 CD 69 D1 18 : 44
:D070 = 0C FE 04 20 05 CD 9F D1 : 70
:D078 = 18 03 CD D2 D1 3E 01 32 : FC

```

33 A2 1A 4C B6 06 6D B1 : 15

さて、仮に、最初のチェックサムが図4-32のようであったとしましょう。まず、トータルサムを見ると違っていますね。つまり、どこかにミスがあるのです。

次にその位置の探し方ですが、縦の和を見ていくと、左から7列目の和が異なっています。これは、ミス位置がこの列内にあることを意味しています。続いて横の和を見ます。上から8行目が違っています。こうして、ミス位置は、次のようにわかります。



ミスの位置が判明したら、指示に従って訂正をして下さい。この例の場合、ミスのあるアドレスは、D03EH番地です。 D03E とキー・インすると、

D03E = DC → ?

↑
現在の内容

と尋ねてきますから、正しい値 C4 を入力して下さい。すると、チェックサムプログラムは再びチェックサムを計算し表示してきます。直っているのを確認したら、リターンキーを押すと、次の128バイトのチェックサムに進みます。

このように「対話型」の方法で、チェックサムがきちんと合うまで操作を続けます。

こうしてすべてが終了すると、正しいプログラムが完成したことになりますから、再びテープにマシン語部分(D000H~D382H)をセーブしておいて下さい。

[注] トータルサム、横の和、縦の和がすべて一致しても、まだミスが残る可能性はあります。しかし、注意深く入力すれば、現実には、このようなことはまず起きませんから御安心下さい。

4-23/ DATA文に直してアスキーセーブ!

さて、この後、BASICのメインルーチンを作ればゲームは完成しますが、このままでは、後日ゲームをする時、まず、BASIC部分をロード、次にマシン語部分をロードという具合に2度手間になってしまいますね。

そこで、DATA文を用いて、マシン語プログラムをBASIC本体内に取り込み、プログラムを1本化することを考えましょう。

しかし、またまた、DATA文として、マシンコードを入力するのはウンザリですね。せっかくチェックサムを合わせたのに、再び入力ミスの可能性も生じてしまいます。そこで、この作業をX1にやってもらうことを考えましょう。

付録4に掲げた「マシン語 DATA ジェネレーター」のプログラムを御覧下さい。このプログラムは指定した範囲のメモリーの内容を自動的に DATA文に取り込んで、行番号つきで生成する「プログラムをつくるプログラム」です。従来のパソコンでも、この種のプログラムはありましたが、X1のHuBASICでは、KEYO文の採用により簡単に作成することができます。KEYO文はこの他にも「ユーザー定義文字作成プログラム」などにも応用される有用な命令ですので、本プログラムを参考に他の応用を考えてみて下さい。

さて、「マシン語 DATA ジェネレーター」が入力セーブを終わり、完成しているいたします。CLEAR & HD000 の後、できあがっているトロンゲームのマシン語部分をロードし、DATAジェネレーターをRUNいたします。指示通りにキーを操作すると、約40秒で、トロンゲームのマシン語部分がDATA文として完成いたします。コンピューターの威力ってすごいですね。（本体プログラムがすでにセーブしてあるとして!）しかる後に、DELETE-900により「DATAジェネレーター」の本体を消去すると、行番号30000から、見事DATA文だけが残りますね。

これをテープにセーブすればよいのですが、後にゲームメイン部分とドッキングさせることを前提に、アスキーセーブというのを試みてみましょう。この言葉、御存知でしたか?

BASICのプログラムがメモリーに格納される時は、省メモリー化のために「中間コード」というものが採用されます。たとえば、GOTO の中間コードは、80H です。何故こんなことをするかというと、GOTO のまま格納すると、4文字ですから4バイト

分メモリーを食います。しかし、中間コードで 80H ならたった1バイトで済んでしまいますね。BASICの各コマンドには、すべてこのような「中間コード」が決められていて、メモリーにはこの形式で格納されており、普通にテープにセーブする時も、この形式でテープに録音されます。

ところが、2つのBASICプログラムを1本に合わせたりする時には、都合の悪いことがおきます。BASICインタプリタは、どこからどこまでが1行で、次の行はどこから始まるかを判断できずに、デタラメなプログラムと見なしてしまうからです。そこで、このような目的のために、アスキーセーブがあるのです。

ただのセーブではなく、アスキー形式でプログラムをセーブすると、プログラムのリスト通り —— つまり「中間コード」を用いずに —— にテープにセーブされるのです。バイト数が多くなりますから、時間がかかりますが、後に2本のプログラムを1本にまとめる（マー
ジするといいます）ためには、是非アスキーセーブをしておいて下さい。

では、どうしたら普通のセーブではなく、アスキーセーブができるのでしょうか。私もそうでしたが、初心者の方はつい見逃がす所だと思います。マニュアル「34ページ」を御覧下さい。ここに、小さく出ているのです（私も最初気づかなかった位です！）。

すなわち、BASICプログラムをアスキーセーブするには、

↓ ファイル名（何でもよい）
SAVE "TRON DATA", A ←
アスキーセーブの指示

とすればよいのです。

まだ御存知なかった方は、是非試してみてください。何回もテープが止まっては動くのを繰り返して、トロンDATAの場合、約1分かかってセーブされるはずですよ。

4-24/トロンゲームマシン語版の完成！

さて、いよいよ長い道のりも大詰めを迎えました。BASICの本体プログラムの作成です。すでに、オールBASIC版は完成しておりますので、少しの変更で済みます。

以下、変更点を述べますので、オールBASIC版リストを参照しながらお読み下さい。

変更点① 初期化ルーチン中、最初の130行でマシン語フリーエリアの確保をしておきます。すなわち、`CLEAR &HD000` を加えます。

図4-33

```

110 / █████ ショキカ █████
120 /
130 CLEAR &HD000 :INIT :WIDTH 40 :CLS 4 :CLICK OFF :TEMPO 200
140 DEFINT A-Z
150 /
160 GOSUB "オフ・ニング"
170 GOSUB "キャラクター・タイキ"

```

変更点② 190行～250行の数値の初期設定は全面的に書き直して、マシン語を書き込むルーチンにします。

図4-34

```

190 / █████ マシン語・サブルーチン / カキコミ █████
200 /
210 RESTORE 30000 :ADR=&HD000
220 FOR I=0 TO &H382
230 READ MC$ :POKE ADR+I,VAL("&H"+MC$)
240 NEXT
250 DEF USR=&HD085 : 'く—— TRON MOVE ルーチン

```

READ文とPOKE文の使い方、よろしいですか？ また、250行でUSR関数の定義をしておきました。

変更点③ 1150行～1280行の出発位置決定ルーチンは、次のように3つのルーチンに分けて書き直してみました。

図4-35

```

1150 / █████ シュツパツ イチ ノ ケツタイ █████
1160 /
1170 RANDOMIZE TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210 U=INT(RND*36)+2
1220 V=INT(RND*19)+3
1230 UNTIL (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DUV=INT(RND*4)*2+2 : 'く—— SARK イトウ・コート (2,4,6,8)
1250 /
1260 / █████ マシン語 ワーク・エリア ショキ セツタイ █████
1270 /
1280 POKE &HE000,X,Y : ' E000H=X, E001H=Y
1290 POKE &HE002,U,V : ' E002H=U, E003H=V
1300 POKE &HE004,4 : ' E004H=DX,DY く—— TRON イトウ・コート (2,4,6,8)
1310 POKE &HE005,DUV : ' E005H=DU,DV く—— SARK イトウ・コート
1320 POKE &HE007,0 : ' E007H=Cont flag (0=Cont, 1=End)
1360 /
1370 / █████ ショキ ヒョウジ █████
1380 /
1500 CGEN 1 :COLOR 7
1510 LOCATE X,Y :PRINT#0 CHR$(120)
1520 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1530 FOR I=1 TO 3 :BEEP :NEXT

```

特に大切なのは、1260行～1320行のマシン語ワークエリアの初期設定の部分です。出発位置をBASICプログラムで決定したら、最初にサークが判断する時のために、ワークエリアに必要なデータを書き込んでおく必要があります。旧BASIC版で使われていたDX, DY, DU, DV という変数は使用せず、移動方向コードに統一します。また、トロンの初期方向は左(コード4)に決めました。

変更点④ 2000行～2550行のメインループは完全に書き換えてしまいます。乱数を書き込み、サーク移動ルーチン(D000H)を呼び出し、続行判定フラグをチェックし、次に、テンキー値をもって、トロンの移動ルーチン(D085H)を呼び出し、続行判定フラグをチェックし、ループに戻るという構成になっています。

図4-36

```

2000 / █████   メイン・ループ   █████
2010 /
2020 / _____ SARK ノ イトウ _____
2030 /
2035 POKE &HE006, INT(RND*100)           :'/<_____ E006H=Random
2040 CALL &HD000                         :'/<_____ SARK MOVE ルーチン
2050 /
2060 F=PEEK(&HE007) :IF F=0 THEN 2110    :'/<_____ Cont flag ノ チェック
2070 T=T+1 :MUSIC "04R3C1DEFGAB+CR3"
2080 GOSUB "カチ ノ ハンタイ"
2090 IF H$("<") THEN 3000 ELSE 1000
2100 /
2110 / _____ TRON ノ イトウ _____
2120 /
2130 I=STICK(0)
2140 A=USR(I)                             :'/ USR=D085H <_____ TRON MOVE ルーチン
2150 /
2160 F=PEEK(&HE007) :IF F=0 THEN 2210    :'/<_____ Cont flag ノ チェック
2170 S=S+1 :MUSIC "02R3+C1BAGFEDCR3"
2180 GOSUB "カチ ノ ハンタイ"
2190 IF H$("<") THEN 3000 ELSE 1000
2200 /
2210 FOR I=0 TO 100 :NEXT :GOTO 2000     :'/ _____> メイン・ループ

```

オールBASIC版に比べ、複雑な条件判断をすべてマシン語ルーチンに任せたため、見やすい構造にできました。また、細かいことですが、効果音を変えてみました(2170行)。これも細かいことですが、乱数を0～99の範囲にしました(2035行)。あと大切な点としては、2210行のFOR～NEXT文です。これは何もしない空ループで、時間かせぎの役割をします。空ループなしでゲームをしてみるとわかりますが、マシン

語により高速になりすぎて、とても人間の反射神経では追いつかない位です。そこで、BASIC版では考えられないことですが、わざと遅くしています。速い方が好きな方は、ループ回数を100より小さく、もう少し遅くしたい方は、100より大きくして、各人に合うスピードに調節して下さい。

変更点⑤ 3020行～3060行の画面反転の部分もマシン語化しましたね。これは次のように画面反転サブルーチン(D36EH)を呼び出せば一発で済みます。

図4-37

```
3000 / █████   ゲーム オフ   █████  
3010 /  
3020 CALL &HD36E      : /<———— 1 ゲーム オン   ルーチン
```

また、これも細かいことですが、旧BASIC版の3100行、3110行の効果音も少し変えてみました。

図4-38

```
3100 IF H$="TRON" THEN MUSIC "05C2DEFGAB+C"  
3110 IF H$="SARK" THEN MUSIC "02+C4BAGFEDC"
```

変更点⑥ 4000行～4360行にわたる移動用サブルーチンは、マシン語部分に吸収しましたから消去いたします。

変更点⑦ 先に、「DATA ジェネレーター」で作成済のマシン語部分を30000行から組み入れます。このためには、DATA文の入っているテープを頭出し状態にしておき、MERGE というBASICコマンドを実行いたします。DATA文の部分は、アスキーセーブしてありましたから、マージが実行されて、1本のプログラムにまとめることができます。

以上が、変更点です。さて、DATA文のマージは完了しましたか？ 細部の体裁をととの

えて、次のようなリストが完成いたしました。テープに一応セーブして、RUNして下さい。

図4-39

```

10 /
20 /
30 / TRON GAME ver.2
40 /
50 / by Yasuhiro Shimizu
60 /
70 / 1983.10.15
80 /
90 /
100 /
110 / ショキカ
120 /
130 CLEAR &HD000 : INIT : WIDTH 40 : CLS 4 : CLICK OFF : TEMPO 200
140 DEFINT A-Z
150 /
160 GOSUB "オープニング"
170 GOSUB "キャラクター・タイキ"
180 /
190 / マシンゴ・ザブ・ル・チン ノ カキコミ
200 /
210 RESTORE 30000 : ADR=&HD000
220 FOR I=0 TO &H382
230 READ MC$ : POKE ADR+I, VAL("&H"+MC$)
240 NEXT
250 DEF USR=&HD085 : /<—— TRON MOVE ル・チン
260 /
270 / セツメイ
280 /
290 CLS
300 LOCATE 10,0 : COLOR 4 : CSIZE 3 : PRINT#0 "TRON GAME" : CSIZE 0
310 LOCATE 13,4 : COLOR 6 : PRINT "TRON:"
320 LOCATE 18,4 : COLOR 7 : CGEN 1 : PRINT CHR$(120) : CGEN 0
330 LOCATE 13,6 : COLOR 2 : PRINT "SARK:"
340 LOCATE 18,6 : COLOR 7 : CGEN 1 : PRINT CHR$(220) : CGEN 0
350 LOCATE 16,8 : PRINT "KEY"
360 LOCATE 17,10 : PRINT "8"
370 LOCATE 16,11 : PRINT "4 6"
380 LOCATE 17,12 : PRINT "2"
390 LOCATE 13,14 : COLOR 3 : PRINT "10 POINT MATCH"
400 LOCATE 13,18 : COLOR 7 : PRINT "HIT"
410 LOCATE 17,18 : CFLASH 1 : PRINT "RETURN KEY" : CFLASH 0
420 REPEAT
430 I$=INKEY$
440 UNTIL I$=CHR$(13)
450 /
460 / ゲーム スタート
470 /
480 T=0 : S=0 : G=0
490 CLS
500 LOCATE 8,8 : COLOR 5 : CSIZE 3
510 PRINT#0 "GAME START" : CSIZE 0
520 MUSIC "04R2C3DEFGAB+C"
530 /
1000 / カメン ツクリ
1010 /
1020 CGEN 0 : CONSOLE 1,24 : CLS : CONSOLE
1030 COLOR 4
1040 LINE ( 0, 1)-(38, 1), "-"
1050 LINE ( 1,23)-(38,23), "-"
1060 LINE ( 0, 2)-( 0,22), "|"
1070 LINE (39, 2)-(39,22), "|"
1080 LOCATE 0, 1 : PRINT "r"
1090 LOCATE 39, 1 : PRINT "r"
1100 LOCATE 0,23 : PRINT "L"
1110 LOCATE 39,23 : PRINT "L"
1120 LOCATE 1, 0 : COLOR 6 : PRINT "TRON:"
1130 LOCATE 20, 0 : COLOR 2 : PRINT "SARK:"
1140 /
1150 / シュワ\°ツ イチ ノ ケツタイ
1160 /
1170 RANDOMIZE TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210 U=INT(RND*36)+2
1220 V=INT(RND*19)+3
1230 UNTIL (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DUV=INT(RND*4)*2+2 : /<—— SARK イト°ウ・コート° (2,4,6,8)
1250 /
1260 / マシンゴ フーク・エリア ショキ セツタイ
1270 /

```

```

1280 POKE &HE000,X,Y          : ' E000H=X, E001H=Y
1290 POKE &HE002,U,V          : ' E002H=U, E003H=V
1300 POKE &HE004,4            : ' E004H=DX,DY <— TRON イトウ・コト (2,4,6,8)
1310 POKE &HE005,DUV          : ' E005H=DU,DV <— SARK イトウ・コト
1320 POKE &HE007,0            : ' E007H=Cont flag (0=Cont, 1=End)
1360 /
1370 /   ■■■■■ ショキ ヒョウシ ■■■■■
1380 /
1500 CGEN 1 :COLOR 7
1510 LOCATE X,Y :PRINT#0 CHR$(120)
1520 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1530 FOR I=1 TO 3 :BEEP :NEXT
1540 /
2000 /   ■■■■■ メイン・ループ ■■■■■
2010 /
2020 /   ——— SARK ノ イトウ ———
2030 /
2035 POKE &HE006,INT(RND*100)  : ' <— E006H=Random
2040 CALL &HD000              : ' <— SARK MOVE ルーチン
2050 /
2060 F=PEEK(&HE007) :IF F=0 THEN 2110 : ' <— Cont flag ノ チェック
2070 T=T+1 :MUSIC "04R3C1DEFGAB+CR3"
2080 GOSUB "カチ ノ ハンタイ"
2090 IF H$(<)" THEN 3000 ELSE 1000
2100 /
2110 /   ——— TRON ノ イトウ ———
2120 /
2130 I=STICK(0)
2140 A=USR(I)                  : ' USR=D005H <— TRON MOVE ルーチン
2150 /
2160 F=PEEK(&HE007) :IF F=0 THEN 2210 : ' <— Cont flag ノ チェック
2170 S=S+1 :MUSIC "02R3+C1BAGFEDCR3"
2180 GOSUB "カチ ノ ハンタイ"
2190 IF H$(<)" THEN 3000 ELSE 1000
2200 /
2210 FOR I=0 TO 100 :NEXT :GOTO 2000 : ' ———> メイン・ループ
2220 /
2290 /   ■■■■■ ゲーム オープン ■■■■■
3000 /
3010 /
3020 CALL &HD36E              : ' <— 1 カメモン ハンテン ルーチン
3070 /
3080 LOCATE 8,8 :COLOR 7 :C$IZE 2
3090 PRINT#0 " GAME!" +H$+" " :C$IZE 0
3100 IF H$="TRON" THEN MUSIC "05C2DEFGAB+C"
3110 IF H$="SARK" THEN MUSIC "02+C4BAGFEDC"
3120 /
3130 LOCATE 5,12 :PRINT "DO YOU WANT REPLAY [ Y or N ]"
3140 REPEAT
3150   I$=INKEY$
3160   UNTIL I$="Y" OR I$="N"
3170 /
3180 IF I$="Y" THEN 460 : ' ———> ゲーム スタート
3190 INIT :CLS
3200 LOCATE 15,10 :PRINT "オツカレサマ ♥"
3210 END
3220 /
3290 /   ※※※※※ BASIC サブ・ルーチン ※※※※※
3990 /
4980 /   ■■■■■
4990 /
5000 /   LABEL "カチ ノ ハンタイ"
5010 /
5020 /   ■■■■■
5030 /
5040 GOSUB "スコア"
5050 IF G=1 THEN 5140
5060 /
5070 H$=""
5080 IF T=10 THEN H$="TRON"
5090 IF S=10 THEN H$="SARK"
5100 RETURN
5110 /
5120 /   ——— シュース ノ ショリ ———
5130 /
5140 H$=""
5150 IF T=S+2 THEN H$="TRON"
5160 IF S=T+2 THEN H$="SARK"
5170 RETURN
5180 /
5190 /
5200 /   ■■■■■
5290 /
6000 /   LABEL "スコア"
6010 /
6020 /   ■■■■■
6030 /
6040 MUSIC "05G1+C" :CGEN 0
6050 LOCATE 8,0 :COLOR 6

```

```

6060 PRINTUSING "##";T;
6070 LOCATE 27,0 :COLOR 2
6080 PRINTUSING "##";S;
6090 /
6100 LOCATE 12,0 :PRINT " ";
6110 IF T>=9 AND S=9 AND T=S THEN G=1 :LOCATE 12,0 :COLOR 7 :CFLASH 1 :PRINT "シュース"; :CFLASH 0
:FOR I=1 TO 2 :BEEP 1 :PAUSE 5 :BEEP 0 :PAUSE 5 :NEXT
6120 /
6130 RETURN
6140 /
6150 /
6160 /
6170 /
6180 /
6190 /
6200 /
6210 /
6220 /
6230 /
6240 /
6250 /
6260 /
6270 /
6280 /
6290 /
6300 /
6310 /
6320 /
6330 /
6340 /
6350 /
6360 /
6370 /
6380 /
6390 /
6400 /
6410 /
6420 /
6430 /
6440 /
6450 /
6460 /
6470 /
6480 /
6490 /
6500 /
6510 /
6520 /
6530 /
6540 /
6550 /
6560 /
6570 /
6580 /
6590 /
6600 /
6610 /
6620 /
6630 /
6640 /
6650 /
6660 /
6670 /
6680 /
6690 /
6700 /
6710 /
6720 /
6730 /
6740 /
6750 /
6760 /
6770 /
6780 /
6790 /
6800 /
6810 /
6820 /
6830 /
6840 /
6850 /
6860 /
6870 /
6880 /
6890 /
6900 /
6910 /
6920 /
6930 /
6940 /
6950 /
6960 /
6970 /
6980 /
6990 /
7000 /
7010 /
7020 /
7030 /
7040 /
7050 /
7060 /
7070 /
7080 /
7090 /
7100 /
7110 /
7120 /
7130 /
7140 /
7150 /
7160 /
7170 /
7180 /
7190 /
7200 /
7210 /
7220 /
7230 /
7240 /
7250 /
7260 /
7270 /
7280 /
7290 /
7300 /
7310 /
7320 /
7330 /
7340 /
7350 /
7360 /
7370 /
7380 /
7390 /
7400 /
7410 /
7420 /
7430 /
7440 /
7450 /
7460 /
7470 /
7480 /
7490 /
7500 /
7510 /
7520 /
7530 /
7540 /
7550 /
7560 /
7570 /
7580 /
7590 /
7600 /
7610 /
7620 /
7630 /
7640 /
7650 /
7660 /
7670 /
7680 /
7690 /
7700 /
7710 /
7720 /
7730 /
7740 /
7750 /
7760 /
7770 /
7780 /
7790 /
7800 /
7810 /
7820 /
7830 /
7840 /
7850 /
7860 /
7870 /
7880 /
7890 /
7900 /
7910 /
7920 /
7930 /
7940 /
7950 /
7960 /
7970 /
7980 /
7990 /
8000 /
8010 /
8020 /
8030 /
8040 /
8050 /
8060 /
8070 /
8080 /
8090 /
8100 /
8110 /
8120 /
8130 /
8140 /
8150 /
8160 /
8170 /
8180 /
8190 /
8200 /
8210 /
8220 /
8230 /
8240 /
8250 /
8260 /
8270 /
8280 /
8290 /
8300 /
8310 /
8320 /
8330 /
8340 /
8350 /
8360 /
8370 /
8380 /
8390 /
8400 /
8410 /
8420 /
8430 /
8440 /
8450 /
8460 /
8470 /
8480 /
8490 /
8500 /
8510 /
8520 /
8530 /
8540 /
8550 /
8560 /
8570 /
8580 /
8590 /
8600 /
8610 /
8620 /
8630 /
8640 /
8650 /
8660 /
8670 /
8680 /
8690 /
8700 /
8710 /
8720 /
8730 /
8740 /
8750 /
8760 /
8770 /
8780 /
8790 /
8800 /
8810 /
8820 /
8830 /
8840 /
8850 /
8860 /
8870 /
8880 /
8890 /
8900 /
8910 /
8920 /
8930 /
8940 /
8950 /
8960 /
8970 /
8980 /
8990 /
9000 /
9010 /
9020 /
9030 /
9040 /
9050 /
9060 /
9070 /
9080 /
9090 /
9100 /
9110 /
9120 /
9130 /
9140 /
9150 /
9160 /
9170 /
9180 /
9190 /
9200 /
9210 /
9220 /
9230 /
9240 /
9250 /
9260 /
9270 /
9280 /
9290 /
9300 /
9310 /
9320 /
9330 /
9340 /
9350 /
9360 /
9370 /
9380 /
9390 /
9400 /
9410 /
9420 /
9430 /
9440 /
9450 /
9460 /
9470 /
9480 /
9490 /
9500 /
9510 /
9520 /
9530 /
9540 /
9550 /
9560 /
9570 /
9580 /
9590 /
9600 /
9610 /
9620 /
9630 /
9640 /
9650 /
9660 /
9670 /
9680 /
9690 /
9700 /
9710 /
9720 /
9730 /
9740 /
9750 /
9760 /
9770 /
9780 /
9790 /
9800 /
9810 /
9820 /
9830 /
9840 /
9850 /
9860 /
9870 /
9880 /
9890 /
9900 /
9910 /
9920 /
9930 /
9940 /
9950 /
9960 /
9970 /
9980 /
9990 /

```

```

30380 DATA E0,11,28,00,B7,ED,52,22
30390 DATA 0A,E0,3E,08,32,0C,E0,3A
30400 DATA 0D,E0,FE,08,20,09,3A,01
30410 DATA E0,3D,32,01,E0,18,07,3A
30420 DATA 03,E0,3D,32,03,E0,37,C9
30430 DATA 2A,08,E0,11,28,08,19,44
30440 DATA 4D,ED,78,FE,20,28,09,B7
30450 DATA C9,2A,08,E0,11,28,08,19
30460 DATA 22,0A,E0,3E,02,32,0C,E0
30470 DATA 3A,0D,E0,FE,00,20,09,3A
30480 DATA 01,E0,3C,32,01,E0,18,07
30490 DATA 3A,03,E0,3C,32,03,E0,37
30500 DATA C9,2A,08,E0,2B,44,4D,ED
30510 DATA 78,FE,20,28,06,B7,C9,2A
30520 DATA 08,E0,2B,22,0A,E0,3E,04
30530 DATA 32,0C,E0,3A,0D,E0,FE,00
30540 DATA 20,09,3A,00,E0,3D,32,00
30550 DATA E0,18,07,3A,02,E0,3D,32
30560 DATA 02,E0,37,C9,2A,08,E0,23
30570 DATA 44,4D,ED,78,FE,20,28,06
30580 DATA B7,C9,2A,08,E0,23,22,0A
30590 DATA E0,3E,06,32,0C,E0,3A,0D
30600 DATA E0,FE,00,20,09,3A,00,E0
30610 DATA 3C,32,08,E0,18,07,3A,02
30620 DATA E0,3C,32,02,E0,37,C9,CD
30630 DATA 26,D2,ED,4B,08,E0,16,00
30640 DATA 3A,0E,E0,5F,CD,3C,D3,3A
30650 DATA 07,E0,FE,00,20,0D,ED,4B
30660 DATA 0A,E0,16,01,3A,0F,E0,5F
30670 DATA C3,3C,D3,ED,4B,0A,E0,16
30680 DATA 02,1E,E8,C3,3C,D3,3A,0C
30690 DATA E0,FE,02,CA,79,D2,FE,04
30700 DATA CA,BA,D2,FE,06,CA,FB,D2
30710 DATA 3A,0D,E0,FE,00,20,05,3A
30720 DATA 04,E0,18,03,3A,05,E0,FE
30730 DATA 04,20,04,3E,99,18,0A,FE
30740 DATA 06,20,04,3E,98,18,02,3E
30750 DATA 91,32,0E,E0,3A,0D,E0,FE
30760 DATA 00,20,08,3E,64,32,0F,E0
30770 DATA 3E,08,32,04,E0,C9,3E,C8
30780 DATA 32,0F,E0,3E,08,32,05,E0
30790 DATA C9,3A,0D,E0,FE,00,20,05
30800 DATA 3A,04,E0,18,03,3A,05,E0
30810 DATA FE,04,20,04,3E,9A,18,0A
30820 DATA FE,06,20,04,3E,97,18,02
30830 DATA 3E,91,32,0E,E0,3A,0D,E0
30840 DATA FE,00,20,08,3E,6E,32,0F
30850 DATA E0,3E,02,32,04,E0,C9,3E
30860 DATA D2,32,0F,E0,3E,02,32,05
30870 DATA E0,C9,3A,0D,E0,FE,00,20
30880 DATA 05,3A,04,E0,18,03,3A,05
30890 DATA E0,FE,08,20,04,3E,97,18
30900 DATA 0A,FE,02,20,04,3E,98,18
30910 DATA 02,3E,90,32,0E,E0,3A,0D
30920 DATA E0,FE,00,20,0B,3E,78,32
30930 DATA 0F,E0,3E,04,32,04,E0,C9
30940 DATA 3E,DC,32,0F,E0,3E,04,32
30950 DATA 05,E0,C9,3A,0D,E0,FE,00
30960 DATA 20,05,3A,04,E0,18,03,3A
30970 DATA 05,E0,FE,08,20,04,3E,9A
30980 DATA 18,0A,FE,02,20,04,3E,99
30990 DATA 18,02,3E,90,32,0E,E0,3A
31000 DATA 0D,E0,FE,00,20,08,3E,02
31010 DATA 32,0F,E0,3E,06,32,04,E0
31020 DATA C9,3E,E6,32,0F,E0,3E,06
31030 DATA 32,05,E0,C9,7A,FE,01,20
31040 DATA 04,3E,27,18,1F,3A,0D,E0
31050 DATA FE,00,20,0D,7A,FE,00,20
31060 DATA 04,3E,26,18,0F,3E,06,18
31070 DATA 0B,7A,FE,00,20,04,3E,22
31080 DATA 18,02,3E,02,CB,A0,ED,79
31090 DATA CB,E0,7B,ED,79,C9,01,00
31100 DATA 20,ED,78,CB,DF,ED,79,03
31110 DATA 78,FE,23,38,F4,79,FE,E8
31120 DATA 38,EF,C9
31130 /
59990 /
60000 / *XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
60010 / *
60020 / * Sample Game for X1 マシン" ニュモン *
60030 / *
60040 / * BASIC & マシン" version *
60050 / *
60060 / *XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

4-25/DISK BASICをお使いの方へ

X1の周辺機器として、フロッピーディスクを購入し、DISK BASIC を使われている方、あるいは新製品X1Dで3インチディスクを使われている方には一言御注意申し上げておくことがあります。

DISK BASIC で、TRONのマシン語版を実行すると、

```
Illegal function call
```

エラーを生じることがあります。これは、プログラム冒頭の130行の

```
CLEAR &HD000
```

が原因です。DISK BASIC では、フリーエリアが減少するために、プログラムの格納場所が不足するのです。このような時は、以下のようにして下さい。

解決法1（安易な方法）

130行の CLEAR 文を

```
CLEAR &HFF00
```

に変更します。すると、フリーエリアが広くなってプログラムは一応動くはずですが。「一応」と述べたのは、この方法が邪道だからです。本文でも強調したように、このようにすると「マシン語部分」が、上からはBASICのテキストに迫られ、下からはスタックに迫られるサンドイッチ領域に置かれてしまうからです。もし、これでエラーを生じた時は次の方法で試みて下さい。

解決法2（プログラム分割法）

DATA文として長々と入っている「マシン語部分」がメモリー不足の原因なので、マシン語部は別にディスクにセーブしておきます。しかる後に、29970行以降のデータ部を

消去します。同時に、190行～240行の「マシン語書き込みルーチン」も消去します
(250行の USR 関数の定義は消してなりません)。130行の CLEAR文は、

CLEAR &HD000

のまま残します。このようにして、実行時はマシン語をロード、続いて BASIC プログラムをロードという2段階で行なえば、TRONゲームは正常に動くはずです。

4-26/最後のコーヒータイム

「トロンゲーム マシン語版」は期待通り動きましたか？ あのモタモタしていた「オール BASIC版」が生き返ったように、スピーディでスリリングなゲームになったと思います。この種のリアルタイムゲームの面白さにとって、スピードがいかに大切であるかを再認識されたことでしょう。

これ位ゲームが速くなると、テンキーをガチャガチャ操作して壊してしまうのでは？ と不安になる程ですね。この時には、ジョイスティックを接続して遊ぶとよいでしょう。プログラム中2130行に `I=STICK(0)` とありますが、これを `I=STICK(1)` とすればジョイスティック1に、`I=STICK(2)` とすればジョイスティック2に変えることができます。マニュアルの154ページを参照して下さい。

以上をもちまして、読者の皆様との共同作業を終わらせていただきます。長い間、おつきあいいただき感謝いたします。

実は、このゲームは私のマシン語ゲーム第1作目でありまして、何かと不備な点もあろうかと思ひます。皆様の創意工夫で改良されて、X1のマシン語をマスターする出発点としていただければ、筆者として最高の喜びです。

私たちは本書を通じて、マシン語によりI/Oポートをアクセスし、代表的な出力装置であるディスプレイテレビの制御法(テキスト画面に限りましたが)をマスターいたしました。また、一つのまとまったマシン語プログラム(ゲーム)を作成することにも成功いたしました。私たちは、マシン語学習における最初の大きな壁を乗り越えたのです！

パソコンテレビX1は、すばらしいハードウェアで、マシン語で制御してみたい部分はまだまだ多く残っています。しかし、これらすべてに解説を加えるには、本書のページ数は余りに少なく、他日の機会を待つことにしたいと思います。

私がX1のマシン語を研究し始めた頃は、余りにもデータが少なく大変な苦勞をいたしましたが、本書が出版される頃には、きっとマイコン雑誌にもX1のマシン語の記事が出るようになっていと思います。本書は、読者の皆様がX1のマシン語をゼロから出発して、基本的な所までマスターできるように考えて執筆されました。雑誌等の記事や、BASICインタプリターの解説を通じて、一步一步次の段階めざして勉強を続けて下さい。

きっと、マシン語の素晴らしさ、さらに言えば、20世紀後半に人類がなしとげた最大の発明の1つと言われている マイクロコンピューター（マイクロプロセッサ） の素晴らしさを身をもって実感されることでしょう。では、またお会いできる時まで……。

■ **あとがき**

■あ と が き

本書はパソコンテレビX1（CZ800C，CZ800D）をZ80マシン語で制御するためのユーザーの立場で書かれた入門書です。最近X1シリーズの新製品X1C，X1Dが発売されました。これらを含めX1シリーズは、いずれもすばらしいハードウェアを持っています。読者の皆様も本書を参考に、X1の機能を100%引き出す方法を考えてみて下さい。X1のハードウェアはきっと皆様の期待に十分応えてくれることでしょう。

本書を書く際に参考にしたZ80マシン語の本を挙げます。これらの本には本書で書ききれなかったテーマも書かれていますから、本書を参考に「X1用」にプログラムを移植してみると力がつくでしょう。

図解マイクロコンピュータ Z-80の使い方（横田英一著 オーム社刊）

この本はZ80CPUの標準的教科書という性格を持っています。

Z80マイコンプログラムテクニック（庄司渉，本田稔著 電波新聞社刊）

各命令が丁寧に解説され、とくにフラグ変化についても細かい説明がなされています。

「マイコンピューター 1982年第7号」 入門・研究特集 Z80アセンブラ言語入門 （CQ出版社刊）

入出力命令について普通の本には書かれていない詳しい説明があります。私はこの本により、X1のI/Oポートの謎を理解できました。また、割り込みについても貴重なプログラム例が出ています。

PC-8001・8801 マシン語入門（塚越一雄著 電波新聞社刊）

主にPC-8001用に書かれていますが、Z80マシン語学習入門書としても使い得る名著。私のマシン語の勉強は、この本との出会いからスタートしました。

PC-8001 マシン語入門Ⅱ（塚越一雄著 電波新聞社刊）

上に挙げた本の続編。アセンブラ指示命令の使い方、USR関数についての説明が出ていま

す。是非「X1用」に翻訳して考えられることをお勧めします。

PC-6001 マシン語入門（桜田幸嗣・田村明史共著 アスキー出版局刊）

乗除算プログラムやソーティングのテクニックなどはX1においても参考になります。

次には、BASICプログラムも含めX1全般について読者の参考となるであろう本・雑誌記事を挙げます。

SHARPパソコンテレビX1 HuBASIC私の勉強ノート（品川ゆり・Dr. Bee著 ラジオ技術社刊）

X1について最も早く出た本。私は、BASICの学習をしていた初期において、とくにサンプルプログラムの解説でプログラミングの力をつけることができました。

パソコンテレビX1 BASIC（戸川隼人著 サイエンス社刊）

マニュアルだけではよくわからない命令の使い方についても丁寧に説明されています。

パソコンテレビライフ（別冊太陽 リビング 平凡社刊）

X1を家庭で使うためのアイデアに富んだ本。カラー写真が大変美しい。

X1テクニカルマスター（ストラットフォードC. C. C. 著 日本ソフトバンク刊）

X1の持つ楽しい機能について易しく説明されています。終わりの方で少しマシン語にも触れていて、グラフィック画面を制御するプログラムが出ています。本書と併読するとよいでしょう。

「ASCII 1983年8月号」 LOAD TEST SHARP X1（アスキー編集部）

ハードウェア、ソフトウェアの詳細が出ていて参考になります。とくにサブCPUとの交信法はこの記事でわかりました。

「I/O 1983年6月号」 X1全回路図（I/O編集部）

メーカー未発表の回路図が編集部の責任において掲載されており、ユーザーにとっては貴重な資料です。

「I/O 1983年3月号」 X1用ミニアセンブラ（dB SOFT）

X1については、著者の知る限り唯一発表されているアセンブラです。小規模のマシン語プログラム作りには威力を発揮します。本書でも一部利用しました。付録5がそのソースリストです。

「マイコン 1983年1月号」 パソコンの互換性を探る（高橋雄一）

X1 HuBASICの中間コード表が出ています。

「マイコン 1983年2月号」 パソコンテレビX1とMZ-700の互換性（高橋雄一）

BASICインタプリターの有効利用にとって不可欠なHuBASICの内部解析結果が出ています。この資料を指針としてBASICを解読すると、時間を節約することができます。

「Oh!MZ 1983年5, 6月号」 X1内部サブルーチンの解析（イッティ・リッターボーン）

BASICインタプリターの初めの方にあるサブルーチンの使い方が述べられています。

「Oh!MZ 1983年7月号」 X1逆アセンブラと命令語の解析（イッティ・リッターボーン）

「マイコン 1983年7月号」 X1ディスアセンブラ（高橋雄一）

上記2つの記事はいずれも、「マシン語→ニーモニック」という逆変換を行なうプログラム（逆アセンブラという）を掲載しています。逆アセンブラはBASICの内部ルーチンの解読には欠かせません。

以上はごく一部です。この他にも多くの雑誌にX1のプログラム（大半がBASICですが）が発表されていますから、参考にして下さい。

最後になりましたが、コンピューターに関して全く無名の私の企画を採用して、本にまとめ上げて下さった日本ソフト&ハード社編集部の皆様ならびにコンピュータイレブン横浜駅西口店の皆様に感謝いたします。

1983年10月

著者記す。

付録 1 Z80 命令表

用いる記号

- n は 8 ビット定数
- nn' は 16 ビット定数, n が上位 8 ビット, n' が下位 8 ビット
(ニーモニックではラベル名を書いてもよい。)
- d は -128 ~ +127 の範囲 (符号付 1 バイト 16 進数) のディスプレイメント。
- e は -128 ~ +127 の範囲 (符号付 1 バイト 16 進数) の相対アドレス。
次の命令の先頭アドレスを 0 とする。
(ニーモニックではラベル名や絶対アドレスを書いてもよい。ただし, アセンブラにより異なる場合もある。)
- Cy は キャリーフラグ
- 添字 H は上位 8 ビット (high), 添字 L は下位 8 ビット (low) を意味する。
- ビット操作命令 SET, RES, BIT で, ビット番号は下記のとおりである。

Diagram illustrating the bit positions of a byte (8 bits):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

左端 (Left End) 右端 (Right End)

↑ 第 7 ビット ↑ 第 0 ビット

- X1では I/O ポートが 64 K バイトあるので、特に次の記号を用いる。
 - I/O (BC) は、BC レジスタペアの内容番地のポートを表わす。
 - I/O (An) は、A レジスタを上位 8 ビット、定数 n を下位 8 ビットとする番地のポートを表わす。
- フラグ変化
 - × 不定
 - 1 1 になる (セットされる)
 - 0 0 になる (リセットされる)
 - ・ 状態にしたがって、セット・リセットされる
 - 変化せず

8 ビットロード命令

| × | A | B | C | D | E | H | L | (HL) | (BC) | (DE) | (IX+d) | (IY+d) | (nn') | n | I | R |
|--------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|------|------|------|----------------|----------------|----------------|---------------------|-----------|-----------|
| LD A, × | 7 F | 7 8 | 7 9 | 7 A | 7 B | 7 C | 7 D | 7 E | 0 A | 1 A | DD 7 E d | FD 7 E d | 3 A n' n | 3 E n | ED 5 7 | ED 5 F |
| LD B, × | 4 7 | 4 0 | 4 1 | 4 2 | 4 3 | 4 4 | 4 5 | 4 6 | | | DD 4 6 d | FD 4 6 d | | 0 6 n | | |
| LD C, × | 4 F | 4 8 | 4 9 | 4 A | 4 B | 4 C | 4 D | 4 E | | | DD 4 E d | FD 4 E d | | 0 E n | | |
| LD D, × | 5 7 | 5 0 | 5 1 | 5 2 | 5 3 | 5 4 | 5 5 | 5 6 | | | DD 5 6 d | FD 5 6 d | | 1 6 n | | |
| LD E, × | 5 F | 5 8 | 5 9 | 5 A | 5 B | 5 C | 5 D | 5 E | | | DD 5 E d | FD 5 E d | | 1 E n | | |
| LD H, × | 6 7 | 6 0 | 6 1 | 6 2 | 6 3 | 6 4 | 6 5 | 6 6 | | | DD 6 6 d | FD 6 6 d | | 2 6 n | | |
| LD L, × | 6 F | 6 8 | 6 9 | 6 A | 6 B | 6 C | 6 D | 6 E | | | DD 6 E d | FD 6 E d | | 2 E n | | |
| LD (HL), × | 7 7 | 7 0 | 7 1 | 7 2 | 7 3 | 7 4 | 7 5 | | | | | | | 3 6 n | | |
| LD (BC), × | 0 2 | | | | | | | | | | | | | | | |
| LD (DE), × | 1 2 | | | | | | | | | | | | | | | |
| LD (IX+d), × | DD 7 7 d | DD 7 0 d | DD 7 1 d | DD 7 2 d | DD 7 3 d | DD 7 4 d | DD 7 5 d | | | | | | | DD 3 6 d n | | |
| LD (IY+d), × | FD 7 7 d | FD 7 0 d | FD 7 1 d | FD 7 2 d | FD 7 3 d | FD 7 4 d | FD 7 5 d | | | | | | | FD 3 6 d n | | |
| LD (nn'), × | 3 2 n' n | | | | | | | | | | | | | | | |
| LD I, × | ED 4 7 | | | | | | | | | | | | | | | |
| LD R, × | ED 4 F | | | | | | | | | | | | | | | |

16 ビットロード命令

| × | AF | BC | DE | HL | SP | IX | IY | nn' | (nn') |
|-------------|-----|-----------------------|-----------------------|----------------|-----------------------|-----------------------|-----------------------|----------------------|-----------------------|
| LD AF, × | | | | | | | | | |
| LD BC, × | | | | | | | | 0 1 n' n | ED 4, B n' n |
| LD DE, × | | | | | | | | 1 1 n' n | ED 5, B n' n |
| LD HL, × | | | | | | | | 2 1 n' n | 2 A n' n |
| LD SP, × | | | | F 9 | | DD F 9 | FD F 9 | 3 1 n' n | ED 7, B n' n |
| LD IX, × | | | | | | | | DD 2 1 n' n | DD 2 A n' n |
| LD IY, × | | | | | | | | FD 2 1 n' n | FD 2 A n' n |
| LD (nn'), × | | ED 4, 3 n' n | ED 5, 3 n' n | 2 2 n' n | ED 7, 3 n' n | DD 2, 2 n' n | FD 2, 2 n' n | | |
| PUSH × | F 5 | C 5 | D 5 | E 5 | | DD E 5 | FD E 5 | | |
| POP × | F 1 | C 1 | D 1 | E 1 | | DD E 1 | FD E 1 | | |

交換・ブロック命令

交換

| | |
|-------------|-----------|
| EX AF, AF | 0 8 |
| EX DE, HL | E B |
| EX (SP), HL | E 3 |
| EX (SP), IX | DD E 3 |
| EX (SP), IY | FD E 3 |
| EXX | D 9 |

ブロック転送

| | |
|------|-----------|
| LDI | ED A 0 |
| LDIR | ED B 0 |
| LDD | ED A 8 |
| LDDR | ED B 8 |

ブロックサーチ

| | |
|------|-----------|
| CPI | ED A 1 |
| CPIR | ED B 1 |
| CPD | ED A 9 |
| CPDR | ED B 9 |

8 ビット 算術・論理演算命令

| × | A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) | n |
|----------|-----|-----|-----|-----|-----|-----|-----|------|----------------|----------------|----------|
| ADD A, × | 8 7 | 8 0 | 8 1 | 8 2 | 8 3 | 8 4 | 8 5 | 8 6 | DD 8 6 d | FD 8 6 d | C 6 n |
| ADC A, × | 8 F | 8 8 | 8 9 | 8 A | 8 B | 8 C | 8 D | 8 E | DD 8 E d | FD 8 E d | C E n |
| SUB × | 9 7 | 9 0 | 9 1 | 9 2 | 9 3 | 9 4 | 9 5 | 9 6 | DD 9 6 d | FD 9 6 d | D 6 n |
| SBC A, × | 9 F | 9 8 | 9 9 | 9 A | 9 B | 9 C | 9 D | 9 E | DD 9 E d | FD 9 E d | D E n |
| AND × | A 7 | A 0 | A 1 | A 2 | A 3 | A 4 | A 5 | A 6 | DD A 6 d | FD A 6 d | E 6 n |
| XOR × | A F | A 8 | A 9 | A A | A B | A C | A D | A E | DD A E d | FD A E d | E E n |
| OR × | B 7 | B 0 | B 1 | B 2 | B 3 | B 4 | B 5 | B 6 | DD B 6 d | FD B 6 d | F 6 n |
| CP × | B F | B 8 | B 9 | B A | B B | B C | B D | B E | DD B E d | FD B E d | F E n |
| INC × | 3 C | 0 4 | 0 C | 1 4 | 1 C | 2 4 | 2 C | 3 4 | DD 3 4 d | FD 3 4 d | |
| DEC × | 3 D | 0 5 | 0 D | 1 5 | 1 D | 2 5 | 2 D | 3 5 | DD 3 5 d | FD 3 5 d | |

16 ビット 算術演算命令

| × | BC | DE | HL | SP | IX | IY |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| ADD HL, × | 0 9 | 1 9 | 2 9 | 3 9 | | |
| ADD IX, × | DD 0 9 | DD 1 9 | | DD 3 9 | DD 2 9 | |
| ADD IY, × | FD 0 9 | FD 1 9 | | FD 3 9 | | FD 2 9 |
| ADC HL, × | ED 4 A | ED 5 A | ED 6 A | ED 7 A | | |
| SBC HL, × | ED 4 2 | ED 5 2 | ED 6 2 | ED 7 2 | | |
| INC × | 0 3 | 1 3 | 2 3 | 3 3 | DD 2 3 | FD 2 3 |
| DEC × | 0 B | 1 B | 2 B | 3 B | DD 2 B | FD 2 B |

ビット操作命令

| × | A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) |
|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-------------------|-------------------|
| BIT 0, × | CB 4 7 | CB 4 0 | CB 4 1 | CB 4 2 | CB 4 3 | CB 4 4 | CB 4 5 | CB 4 6 | DD CB d 4 6 | FD CB d 4 6 |
| BIT 1, × | CB 4 F | CB 4 8 | CB 4 9 | CB 4 A | CB 4 B | CB 4 C | CB 4 D | CB 4 E | DD CB d 4 E | FD CB d 4 E |
| BIT 2, × | CB 5 7 | CB 5 0 | CB 5 1 | CB 5 2 | CB 5 3 | CB 5 4 | CB 5 5 | CB 5 6 | DD CB d 5 6 | FD CB d 5 6 |
| BIT 3, × | CB 5 F | CB 5 8 | CB 5 9 | CB 5 A | CB 5 B | CB 5 C | CB 5 D | CB 5 E | DD CB d 5 E | FD CB d 5 E |
| BIT 4, × | CB 6 7 | CB 6 0 | CB 6 1 | CB 6 2 | CB 6 3 | CB 6 4 | CB 6 5 | CB 6 6 | DD CB d 6 6 | FD CB d 6 6 |
| BIT 5, × | CB 6 F | CB 6 8 | CB 6 9 | CB 6 A | CB 6 B | CB 6 C | CB 6 D | CB 6 E | DD CB d 6 E | FD CB d 6 E |
| BIT 6, × | CB 7 7 | CB 7 0 | CB 7 1 | CB 7 2 | CB 7 3 | CB 7 4 | CB 7 5 | CB 7 6 | DD CB d 7 6 | FD CB d 7 6 |
| BIT 7, × | CB 7 F | CB 7 8 | CB 7 9 | CB 7 A | CB 7 B | CB 7 C | CB 7 D | CB 7 E | DD CB d 7 E | FD CB d 7 E |
| RES 0, × | CB 8 7 | CB 8 0 | CB 8 1 | CB 8 2 | CB 8 3 | CB 8 4 | CB 8 5 | CB 8 6 | DD CB d 8 6 | FD CB d 8 6 |
| RES 1, × | CB 8 F | CB 8 8 | CB 8 9 | CB 8 A | CB 8 B | CB 8 C | CB 8 D | CB 8 E | DD CB d 8 E | FD CB d 8 E |
| RES 2, × | CB 9 7 | CB 9 0 | CB 9 1 | CB 9 2 | CB 9 3 | CB 9 4 | CB 9 5 | CB 9 6 | DD CB d 9 6 | FD CB d 9 6 |
| RES 3, × | CB 9 F | CB 9 8 | CB 9 9 | CB 9 A | CB 9 B | CB 9 C | CB 9 D | CB 9 E | DD CB d 9 E | FD CB d 9 E |
| RES 4, × | CB A 7 | CB A 0 | CB A 1 | CB A 2 | CB A 3 | CB A 4 | CB A 5 | CB A 6 | DD CB d A 6 | FD CB d A 6 |
| RES 5, × | CB A F | CB A 8 | CB A 9 | CB A A | CB A B | CB A C | CB A D | CB A E | DD CB d A E | FD CB d A E |
| RES 6, × | CB B 7 | CB B 0 | CB B 1 | CB B 2 | CB B 3 | CB B 4 | CB B 5 | CB B 6 | DD CB d B 6 | FD CB d B 6 |
| RES 7, × | CB B F | CB B 8 | CB B 9 | CB B A | CB B B | CB B C | CB B D | CB B E | DD CB d B E | FD CB d B E |
| SET 0, × | CB C 7 | CB C 0 | CB C 1 | CB C 2 | CB C 3 | CB C 4 | CB C 5 | CB C 6 | DD CB d C 6 | FD CB d C 6 |
| SET 1, × | CB C F | CB C 8 | CB C 9 | CB C A | CB C B | CB C C | CB C D | CB C E | DD CB d C E | FD CB d C E |
| SET 2, × | CB D 7 | CB D 0 | CB D 1 | CB D 2 | CB D 3 | CB D 4 | CB D 5 | CB D 6 | DD CB d D 6 | FD CB d D 6 |
| SET 3, × | CB D F | CB D 8 | CB D 9 | CB D A | CB D B | CB D C | CB D D | CB D E | DD CB d D E | FD CB d D E |
| SET 4, × | CB E 7 | CB E 0 | CB E 1 | CB E 2 | CB E 3 | CB E 4 | CB E 5 | CB E 6 | DD CB d E 6 | FD CB d E 6 |
| SET 5, × | CB E F | CB E 8 | CB E 9 | CB E A | CB E B | CB E C | CB E D | CB E E | DD CB d E E | FD CB d E E |
| SET 6, × | CB F 7 | CB F 0 | CB F 1 | CB F 2 | CB F 3 | CB F 4 | CB F 5 | CB F 6 | DD CB d F 6 | FD CB d F 6 |
| SET 7, × | CB F F | CB F 8 | CB F 9 | CB F A | CB F B | CB F C | CB F D | CB F E | DD CB d F E | FD CB d F E |

アキュムレータ・フラグ・CPU制御命令

アキュムレータ・フラグ制御

| | |
|-----|-----------|
| DAA | 2 7 |
| CPL | 2 F |
| NEG | ED 4 4 |
| CCF | 3 F |
| SCF | 3 7 |

CPU 制御

| | |
|------|-----------|
| NOP | 0 0 |
| HALT | 7 6 |
| DI | F 3 |
| EI | F B |
| IM 0 | ED 4 6 |
| IM 1 | ED 5 6 |
| IM 2 | ED 5 E |

ローテート・シフト命令

| × | A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|----------------------|----------------------|
| RLC × | CB 0 7 | CB 0 0 | CB 0 1 | CB 0 2 | CB 0 3 | CB 0 4 | CB 0 5 | CB 0 6 | DD CB d 0 6 | FD CB d 0 6 |
| RRC × | CB 0 F | CB 0 8 | CB 0 9 | CB 0 A | CB 0 B | CB 0 C | CB 0 D | CB 0 E | DD CB d 0 E | FD CB d 0 E |
| RL × | CB 1 7 | CB 1 0 | CB 1 1 | CB 1 2 | CB 1 3 | CB 1 4 | CB 1 5 | CB 1 6 | DD CB d 1 6 | FD CB d 1 6 |
| RR × | CB 1 F | CB 1 8 | CB 1 9 | CB 1 A | CB 1 B | CB 1 C | CB 1 D | CB 1 E | DD CB d 1 E | FD CB d 1 E |
| SLA × | CB 2 7 | CB 2 0 | CB 2 1 | CB 2 2 | CB 2 3 | CB 2 4 | CB 2 5 | CB 2 6 | DD CB d 2 6 | DD CB d 2 6 |
| SRA × | CB 2 F | CB 2 8 | CB 2 9 | CB 2 A | CB 2 B | CB 2 C | CB 2 D | CB 2 E | DD CB d 2 E | FD CB d 2 E |
| SRL × | CB 3 F | CB 3 8 | CB 3 9 | CB 3 A | CB 3 B | CB 3 C | CB 3 D | CB 3 E | DD CB d 3 E | FD CB d 3 E |
| RLD | | | | | | | | ED 6 F | | |
| RRD | | | | | | | | ED 6 7 | | |

| | |
|------|-----|
| RLCA | 0 7 |
| RRCA | 0 F |
| RLA | 1 7 |
| RRA | 1 F |

ジャンプ命令

| × | 無条件 | キャリー | ノン キャリー | ゼロ | ノン ゼロ | パリティ 偶数 | パリティ 奇数 | 負 | 正 | カウント |
|-----------|----------------|---------------|----------------|---------------|----------------|---------------|----------------|---------------|----------------|----------|
| | | C | NC | Z | NZ | PE | PO | M | P | |
| JP ×, nn' | C 3 n' n | DA n' n | D 2 n' n | CA n' n | C 2 n' n | EA n' n | E 2 n' n | FA n' n | F 2 n' n | |
| JR ×, e | 1 8 e | 3 8 e | 3 0 e | 2 8 e | 2 0 e | | | | | |
| JP (HL) | E 9 | | | | | | | | | |
| JP (IX) | DD E 9 | | | | | | | | | |
| JP (IY) | FD E 9 | | | | | | | | | |
| DJNZ e | | | | | | | | | | 1 0 e |

コール・リターン命令

コール・リターン

| × | 無条件 | キャリー | ノン キャリー | ゼロ | ノン ゼロ | パリティ 偶数 | パリティ 奇数 | 負 | 正 |
|-------------|---------------|---------------|----------------|---------------|----------------|---------------|----------------|---------------|----------------|
| | | C | NC | Z | NZ | PE | PO | M | P |
| CALL ×, nn' | CD n' n | DC n' n | D 4 n' n | CC n' n | C 4 n' n | EC n' n | E 4 n' n | FC n' n | F 4 n' n |
| RET × | C 9 | D 8 | D 0 | C 8 | C 0 | E 8 | E 0 | F 8 | F 0 |
| RETI | ED 4 D | | | | | | | | |
| RETN | ED 4 5 | | | | | | | | |

リスタート

| × | 00H | 08H | 10H | 18H | 20H | 28H | 30H | 38H |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| RST × | C 7 | CF | D 7 | DF | E 7 | EF | F 7 | FF |

入出力命令

入力

| \times | A | B | C | D | E | H | L |
|-------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| IN \times , (n) | DB n | | | | | | |
| IN \times , (C) | ED 7 8 | ED 4 0 | ED 4 8 | ED 5 0 | ED 5 8 | ED 6 0 | ED 6 8 |

| | | |
|------|-----------|--------------|
| INI | ED A 2 | } ブロック入力コマンド |
| INIR | ED B 2 | |
| IND | ED A A | |
| INDR | ED B A | |

出力

| \times | A | B | C | D | E | H | L |
|-------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| OUT (n), \times | D 3 n | | | | | | |
| OUT (C), \times | ED 7 9 | ED 4 1 | ED 4 9 | ED 5 1 | ED 5 9 | ED 6 1 | ED 6 9 |

| | | |
|-------|-----------|--------------|
| OUT I | ED A 3 | } ブロック出力コマンド |
| OTIR | ED B 3 | |
| OUTD | ED A B | |
| OTDR | ED B B | |

付 録

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|---------------|----------------|-------|---|---|-----|---|---|---|------------------------|--|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| ADC A, n | CE <u>ⓓ</u> | | | | | | | | 7 | 8ビット加算キャリ付 (アド・ウィズ・キャリ) $A \leftarrow A + \text{ソース} + C_y$ |
| ADC A, A | 8F | | | | | | | | 4 | |
| ADC A, B | 88 | | | | | | | | | |
| ADC A, C | 89 | | | | | | | | | |
| ADC A, D | 8A | | | | | | | | | |
| ADC A, E | 8B | • | • | • | — | • | 0 | • | 4 | |
| ADC A, H | 8C | | | | | | | | | |
| ADC A, L | 8D | | | | | | | | | |
| ADC A, (HL) | 8E | | | | | | | | | |
| ADC A, (IX+d) | DD 8E <u>ⓓ</u> | | | | | | | | 19 | |
| ADC A, (IY+d) | FD 8E <u>ⓓ</u> | | | | | | | | 19 | |
| ADC HL, BC | ED 4A | | | | | | | | 15 | 16ビット加算キャリ付 (アド・ウィズ・キャリ) $HL \leftarrow HL + \text{ソース} + C_y$ |
| ADC HL, DE | ED 5A | • | • | × | — | • | 0 | • | | |
| ADC HL, HL | ED 6A | | | | | | | | | |
| ADC HL, SP | ED 7A | | | | | | | | | |
| ADD A, n | C6 <u>ⓓ</u> | | | | | | | | 7 | 8ビット加算 (アド) $A \leftarrow A + \text{ソース}$ |
| ADD A, A | 87 | | | | | | | | 4 | |
| ADD A, B | 80 | | | | | | | | | |
| ADD A, C | 81 | | | | | | | | | |
| ADD A, D | 82 | | | | | | | | | |
| ADD A, E | 83 | • | • | • | — | • | 0 | • | 4 | |
| ADD A, H | 84 | | | | | | | | | |
| ADD A, L | 85 | | | | | | | | | |
| ADD A, (HL) | 86 | | | | | | | | | |
| ADD A, (IX+d) | DD 86 <u>ⓓ</u> | | | | | | | | 19 | |
| ADD A, (IY+d) | FD 86 <u>ⓓ</u> | | | | | | | | 19 | |
| ADD HL, BC | 09 | | | | | | | | 11 | 16ビット加算 (アド) $HL \leftarrow HL + \text{ソース}$ |
| ADD HL, DE | 19 | | | | | | | | | |
| ADD HL, HL | 29 | | | | | | | | | |
| ADD HL, SP | 39 | | | | | | | | | |
| ADD IX, BC | DD 09 | | | | | | | | 15 | $IX \leftarrow IX + \text{ソース}$ |
| ADD IX, DE | DD 19 | — | — | × | — | — | 0 | • | | |
| ADD IX, IX | DD 29 | | | | | | | | | |
| ADD IX, SP | DD 39 | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | | | | | | | | | | | | | | | | |
|---|--|-------|---|---|-----|---|---|---|--|---|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S | Z | H | P/V | | N | C | | | | | | | | | | | | | | | | | | |
| | | | | | P | V | | | | | | | | | | | | | | | | | | | | |
| ADD IY, BC ADD IY, DE ADD IY, IY ADD IY, SP | FD 09 FD 19 FD 29 FD 39 | | | | | | | | | } IY← IY+ソース | | | | | | | | | | | | | | | | |
| AND n AND A AND B AND C AND D AND E AND H AND L AND (HL) AND (IX+d) AND (IY+d) | E6 <u>n</u> A7 A0 A1 A2 A3 A4 A5 A6 DD A6 <u>d</u> FD A6 <u>d</u> | | | | | | | | 7 7 19 19 | | 論理積 (アンド) A←A∧ソース <table><tr><th>a</th><th>b</th><th>答</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | a | b | 答 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| a | b | 答 | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| BIT 0, A BIT 0, B BIT 0, C BIT 0, D BIT 0, E BIT 0, H BIT 0, L BIT 0, (HL) BIT 0, (IX+d) BIT 0, (IY+d) | CB 47 CB 40 CB 41 CB 42 CB 43 CB 44 CB 45 CB 46 DD CB <u>d</u> 46 FD CB <u>d</u> 46 | | | | | | | | | } 8 12 20 20 | ビットテスト ソースの第0ビットを調べ Zフラグを設定 | | | | | | | | | | | | | | | |
| BIT 1, A BIT 1, B BIT 1, C BIT 1, D BIT 1, E BIT 1, H BIT 1, L BIT 1, (HL) BIT 1, (IX+d) BIT 1, (IY+d) | CB 4F CB 48 CB 49 CB 4A CB 4B CB 4C CB 4D CB 4E DD CB <u>d</u> 4E FD CB <u>d</u> 4E | | | | | | | | | | | | | | | | | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|---------------|--------------------|-------|---|---|-----|---|---|---|------------------------|---------------------------------------|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| BIT 2, A | CB 57 | | | | | | | | } 8 | ビットテスト ソースの第2ビットを調べ Zフラグを設定 |
| BIT 2, B | CB 50 | | | | | | | | | |
| BIT 2, C | CB 51 | | | | | | | | | |
| BIT 2, D | CB 52 | | | | | | | | | |
| BIT 2, E | CB 53 | | | | | | | | | |
| BIT 2, H | CB 54 | × | • | 1 | × | — | 0 | — | | |
| BIT 2, L | CB 55 | | | | | | | | | |
| BIT 2, (HL) | CB 56 | | | | | | | | | |
| BIT 2, (IX+d) | DD CB <u>d</u> ,56 | | | | | | | | 20 | |
| BIT 2, (IY+d) | FD CB <u>d</u> ,56 | | | | | | | | 20 | |
| BIT 3, A | CB 5F | | | | | | | | } 8 | ビットテスト ソースの第3ビットを調べ Zフラグを設定 |
| BIT 3, B | CB 58 | | | | | | | | | |
| BIT 3, C | CB 59 | | | | | | | | | |
| BIT 3, D | CB 5A | | | | | | | | | |
| BIT 3, E | CB 5B | | | | | | | | | |
| BIT 3, H | CB 5C | × | • | 1 | × | — | 0 | — | | |
| BIT 3, L | CB 5D | | | | | | | | | |
| BIT 3, (HL) | CB 5E | | | | | | | | | |
| BIT 3, (IX+d) | DD CB <u>d</u> ,5E | | | | | | | | 20 | |
| BIT 3, (IY+d) | FD CB <u>d</u> ,5E | | | | | | | | 20 | |
| BIT 4, A | CB 67 | | | | | | | | } 8 | ビットテスト ソースの第4ビットを調べ Zフラグを設定 |
| BIT 4, B | CB 60 | | | | | | | | | |
| BIT 4, C | CB 61 | | | | | | | | | |
| BIT 4, D | CB 62 | | | | | | | | | |
| BIT 4, E | CB 63 | | | | | | | | | |
| BIT 4, H | CB 64 | × | • | 1 | × | — | 0 | — | | |
| BIT 4, L | CB 65 | | | | | | | | | |
| BIT 4, (HL) | CB 66 | | | | | | | | | |
| BIT 4, (IX+d) | DD CB <u>d</u> ,66 | | | | | | | | 20 | |
| BIT 4, (IY+d) | FD CB <u>d</u> ,66 | | | | | | | | 20 | |
| BIT 5, A | CB 6F | | | | | | | | } 8 | ビットテスト ソースの第5ビットを調べ Zフラグを設定 |
| BIT 5, B | CB 68 | | | | | | | | | |
| BIT 5, C | CB 69 | | | | | | | | | |
| BIT 5, D | CB 6A | | | | | | | | | |
| BIT 5, E | CB 6B | × | • | 1 | × | — | 0 | — | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|---------------|-------------------|-------|---|---|-----|---|---|---|------------------------|---|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| BIT 5, H | CB 6C | | | | | | | | 12 20 20 | |
| BIT 5, L | CB 6D | | | | | | | | | |
| BIT 5, (HL) | CB 6E | | | | | | | | | |
| BIT 5, (IX+d) | DD CB <u>d</u> 6E | | | | | | | | | |
| BIT 5, (IY+d) | FD CB <u>d</u> 6E | | | | | | | | | |
| BIT 6, A | CB 77 | | | | | | | | 8 12 20 20 | ビットテスト ソースの第6ビットを調べ Zフラグを設定 |
| BIT 6, B | CB 70 | | | | | | | | | |
| BIT 6, C | CB 71 | | | | | | | | | |
| BIT 6, D | CB 72 | | | | | | | | | |
| BIT 6, E | CB 73 | × | • | 1 | × | — | 0 | — | | |
| BIT 6, H | CB 74 | | | | | | | | | |
| BIT 6, L | CB 75 | | | | | | | | | |
| BIT 6, (HL) | CB 76 | | | | | | | | | |
| BIT 6, (IX+d) | DD CB <u>d</u> 76 | | | | | | | | | |
| BIT 6, (IY+d) | FD CB <u>d</u> 76 | | | | | | | | | |
| BIT 7, A | CB 7F | | | | | | | | 8 12 20 20 | ビットテスト ソースの第7ビットを調べ Zフラグを設定 |
| BIT 7, B | CB 78 | | | | | | | | | |
| BIT 7, C | CB 79 | | | | | | | | | |
| BIT 7, D | CB 7A | | | | | | | | | |
| BIT 7, E | CB 7B | | | | | | | | | |
| BIT 7, H | CB 7C | × | • | 1 | × | — | 0 | — | | |
| BIT 7, L | CB 7D | | | | | | | | | |
| BIT 7, (HL) | CB 7E | | | | | | | | | |
| BIT 7, (IX+d) | DD CB <u>d</u> 7E | | | | | | | | | |
| BIT 7, (IY+d) | FD CB <u>d</u> 7E | | | | | | | | | |
| CALL NZ, nn' | C4 <u>nn'</u> | | | | | | | | 成立時 17 不成立 10 | サブルーチン・コール (条件付) ・条件が成立すれば戻り番地〔PC〕をスタックへPUSHし nn'へジャンプ〔PC←nn'〕 ・成立しなければ本命令は無視する |
| CALL Z, nn' | CC <u>nn'</u> | | | | | | | | | |
| CALL NC, nn' | D4 <u>nn'</u> | | | | | | | | | |
| CALL C, nn' | DC <u>nn'</u> | — | — | — | — | — | — | — | | |
| CALL PO, nn' | E4 <u>nn'</u> | | | | | | | | | |
| CALL PE, nn' | EC <u>nn'</u> | | | | | | | | | |
| CALL P, nn' | F4 <u>nn'</u> | | | | | | | | | |
| CALL M, nn' | FC <u>nn'</u> | | | | | | | | | |
| CALL nn' | CD <u>nn'</u> | — | — | — | — | — | — | — | 17 | サブルーチン・コール (無条件) PCをスタックへPUSHしPC←nn' |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|-----------|----------------|-------|---|---|-----|---|---|---|---------------------------------|---|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| CCF | 3F | — | — | × | — | — | 0 | • | 4 | Cy を反転〔Cy ← $\overline{\text{Cy}}$ 〕 |
| CP n | FE <u>n</u> | | | | | | | | 7 | 比較 (コンペア) A—ソースの演算をする Aの内容は変わらずフラグ だけが変化する |
| CP A | BF | | | | | | | | 4 | |
| CP B | B8 | | | | | | | | | |
| CP C | B9 | | | | | | | | | |
| CP D | BA | | | | | | | | | |
| CP E | BB | • | • | • | — | • | 1 | • | | |
| CP H | BC | | | | | | | | | |
| CP L | BD | | | | | | | | | |
| CP (HL) | BE | | | | | | | | 7 | |
| CP (IX+d) | DD BE <u>d</u> | | | | | | | | 19 | |
| CP (IY+d) | FD BE <u>d</u> | | | | | | | | 19 | |
| CPD | ED A9 | × | • | × | — | • | 1 | — | 16 | 比較 (コンペア・ディクリメント) A—(HL)のフラグ変化のみ HL←HL−1 BC←BC−1 |
| CPDR | ED B9 | × | • | × | — | • | 1 | — | 1バイト につき 21 最終のみ 16 | 比較 (コンペア・ディクリメント・ リピート) CPDをA=(HL)〔Zフラグ=1〕 またはBC=0〔Vフラグ=0〕 までくり返す |
| CPI | ED A1 | × | • | × | — | • | 1 | — | 16 | 比較 (コンペア・インクリメント) A—(HL)のフラグ変化のみ HL←HL+1 BC←BC−1 |
| CPIR | ED B1 | × | • | × | — | • | 1 | — | 1バイト につき 21 最終のみ 16 | 比較 (コンペア・インクリメント・ リピート) CPIをA=(HL)〔Zフラグ=1〕 またはBC=0〔Vフラグ=0〕 までくり返す |
| CPL | 2F | — | — | 1 | — | — | 1 | — | 4 | コンプリメント Aレジスタに対しビット反転 $0 \rightarrow 1$ $1 \rightarrow 0$ |
| DAA | 27 | • | • | • | • | — | — | • | 4 | デジマル・アジャスト・アキュムレーター Aレジスタに対し10進補正 |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|-------------|----------------|-------|---|---|-----|---|---|---|---------------------------|---|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| DEC A | 3D | | | | | | | | 4 | 8ビットディクリメント ソース←ソース-1 |
| DEC B | 05 | | | | | | | | | |
| DEC C | 0D | | | | | | | | | |
| DEC D | 15 | | | | | | | | | |
| DEC E | 1D | | | | | | | | 11 | |
| DEC H | 25 | • | • | • | - | • | 1 | - | | |
| DEC L | 2D | | | | | | | | | |
| DEC (HL) | 35 | | | | | | | | | |
| DEC (IX+d) | DD 35 <u>d</u> | | | | | | | | 23 | |
| DEC (IY+d) | FD 35 <u>d</u> | | | | | | | | 23 | |
| DEC BC | 0B | | | | | | | | 6 | 16ビットディクリメント ソース←ソース-1 |
| DEC DE | 1B | | | | | | | | | |
| DEC HL | 2B | - | - | - | - | - | - | - | | |
| DEC SP | 3B | - | - | - | - | - | - | - | | |
| DEC IX | DD 2B | | | | | | | | 10 | |
| DEC IY | FD 2B | | | | | | | | 10 | |
| DI | F3 | - | - | - | - | - | - | - | 4 | 割り込み禁止 (ディセーブル・ インタラプト) |
| DJNZ e | 10 <u>e</u> | | | | | | | | B = 0 8 B ≠ 0 13 | (ディクリメント・ジャンプノンゼロ) B←B-1 B≠0ならeバイトだ けジャンプ〔PC←PC+e〕 B=0ならジャンプせず〔e=0と同じ〕 |
| EI | FB | - | - | - | - | - | - | - | 4 | 割り込み許可 (インネーブル・ インタラプト) |
| EX (SP), HL | E3 | | | | | | | | 19 | 交換 (エクスチェンジ) ソースとディスティネーション の内容を交換する |
| EX (SP), IX | DD E3 | | | | | | | | 23 | |
| EX (SP), IY | FD E3 | - | - | - | - | - | - | - | 23 | |
| EX AF, AF' | 08 | | | | | | | | 4 | |
| EX DE, HL | EB | | | | | | | | 4 | |
| EXX | D9 | - | - | - | - | - | - | - | 4 | BC DE HLの内容と BC' DE' HL'の内容を交換する |
| HALT | 76 | - | - | - | - | - | - | - | 4 | 命令実行の進行を止めリセット または割り込み待ちとなる (ホルト) |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所要 クロック サイクル | 動作 |
|------------|----------------|-------|---|---|-----|---|---|---|--------------------|--|
| | | S | Z | H | P/V | | N | C | | |
| IM 0 | ED 46 | | | | | | | | 8 | 割り込みモードを0に設定する |
| IM 1 | ED 56 | - | - | - | - | - | - | - | 8 | 割り込みモードを1に設定する |
| IM 2 | ED 5E | | | | | | | | 8 | 割り込みモードを2に設定する |
| INC A | 3C | | | | | | | | 4 | 8ビットインクリメント ソース←ソース+1 |
| INC B | 04 | | | | | | | | | |
| INC C | 0C | | | | | | | | | |
| INC D | 14 | | | | | | | | | |
| INC E | 1C | | | | | | | | 11 | |
| INC H | 24 | • | • | • | - | • | 0 | - | | |
| INC L | 2C | | | | | | | | | |
| INC (HL) | 34 | | | | | | | | | |
| INC (IX+d) | DD 34 <u>d</u> | | | | | | | | 23 | |
| INC (IY+d) | FD 34 <u>d</u> | | | | | | | | 23 | |
| INC BC | 03 | | | | | | | | 6 | 16ビットインクリメント ソース←ソース+1 |
| INC DE | 13 | | | | | | | | | |
| INC HL | 23 | - | - | - | - | - | - | - | | |
| INC SP | 33 | | | | | | | | | |
| INC IX | DD 23 | | | | | | | | 10 | |
| INC IY | FD 23 | | | | | | | | 10 | |
| IN A, (C) | ED 78 | | | | | | | | 12 | 入力 BCレジスタの内容番地のポートからディスティネーションのレジスタへ入力 〔ディスティネーション←I/O (BC)〕 |
| IN B, (C) | ED 40 | | | | | | | | | |
| IN C, (C) | ED 48 | | | | | | | | | |
| IN D, (C) | ED 50 | • | • | 0 | • | - | 0 | - | | |
| IN E, (C) | ED 58 | | | | | | | | 11 | |
| IN H, (C) | ED 60 | | | | | | | | | |
| IN L, (C) | ED 68 | | | | | | | | | |
| IN A, (n) | DB <u>n</u> | - | - | - | - | - | - | - | | |
| IND | ED AA | × | • | × | × | - | × | × | 16 | イン・ディクリメント (HL) ← I/O (BC), HL ← HL - 1 B ← B - 1 |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|------------|---------------|-------|---|---|-----|---|---|---|---------------------------------|--|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| INDR | ED BA | × | 1 | × | × | — | × | × | 1バイト につき 21 最終のみ 16 | イン・ディクリメント・リピート INDをB=0までくり返す |
| INI | ED A2 | × | • | × | × | — | × | × | 16 | イン・インクリメント (HL)←I/0(BC), HL←HL+1 B←B-1 |
| INIR | ED B2 | × | 1 | × | × | — | × | × | 1バイト につき 21 最終のみ 16 | イン・インクリメント・リピート INIをB=0までくり返す |
| JP (HL) | E9 | | | | | | | | 4 | } ジャンプ 各レジスタの内容番地へジャンプ 〔PC←HLなど〕 nn'番地へジャンプ〔PC←nn'〕 |
| JP (IX) | DD E9 | — | — | — | — | — | — | — | 8 | |
| JP (IY) | FD E9 | | | | | | | | 8 | |
| JP nn' | C3 <u>nn'</u> | | | | | | | | 10 | |
| JP NZ, nn' | C2 <u>nn'</u> | | | | | | | | 10 | ジャンプ (条件付) ・条件が成立すれば nn'番地へ ジャンプ (PC←nn') ・不成立なら本命令は無視する |
| JP Z, nn' | CA <u>nn'</u> | | | | | | | | | |
| JP NC, nn' | D2 <u>nn'</u> | | | | | | | | | |
| JP C, nn' | DA <u>nn'</u> | — | — | — | — | — | — | — | | |
| JP PO, nn' | E2 <u>nn'</u> | | | | | | | | | |
| JP PE, nn' | EA <u>nn'</u> | | | | | | | | | |
| JP P, nn' | F2 <u>nn'</u> | | | | | | | | | |
| JP M, nn' | FA <u>nn'</u> | | | | | | | | | |
| JR e | 18 <u>e</u> | — | — | — | — | — | — | — | 12 | ジャンプ・リラティブ (無条件) eバイト先へジャンプする 〔PC←PC+e〕 |
| JR NZ, e | 20 <u>e</u> | | | | | | | | 条 件 成 立 12 条 件 | ジャンプ・リラティブ (条件付) ・条件が成立すれば eバイト先へ ジャンプ 〔PC←PC+e〕 |
| JR Z, e | 28 <u>e</u> | — | — | — | — | — | — | — | | |
| JR NC, e | 30 <u>e</u> | | | | | | | | | |
| JR C, e | 38 <u>e</u> | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所要 クロック サイクル | 動作 |
|---|--|-------|---|---|-----|---|---|---|---|----------------|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| | | | | | | | | | 不 成 立 7 | ・不成立なら本命令は無視する |
| LD A, n LD A, A LD A, B LD A, C LD A, D LD A, E LD A, H LD A, L LD A, (nn') LD A, (BC) LD A, (DE) LD A, (HL) LD A, (IX+d) LD A, (IY+d) | 3E <u>n</u> 7F 78 79 7A 7B 7C 7D 3A <u>n'</u> <u>n</u> 0A 1A 7E DD 7E <u>d</u> FD 7E <u>d</u> | | | | | | | | 7 | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 | |
|--------------|----------------|-------|---|---|-----|---|---|---------------------|---------------------------|---|
| | | S | Z | H | P/V | | N | | | C |
| | | | | | P | V | | | | |
| LD C, n | 0E <u>n</u> | | | | | | | 7 | 8ビット転送 (ロード) C←ソース | |
| LD C, A | 4F | | | | | | | } 4 | | |
| LD C, B | 48 | | | | | | | | | |
| LD C, C | 49 | | | | | | | | | |
| LD C, D | 4A | | | | | | | | | |
| LD C, E | 4B | - | - | - | - | - | - | | | |
| LD C, H | 4C | | | | | | | } 4 | | |
| LD C, L | 4D | | | | | | | | | |
| LD C, (HL) | 4E | | | | | | | | | |
| LD C, (IX+d) | DD 4E <u>d</u> | | | | | | | 7 | | |
| | | | | | | | | 19 | | |
| LD C, (IY+d) | FD 4E <u>d</u> | | | | | | | 19 | | |
| | | | | | | | | | | |
| LD D, n | 16 <u>n</u> | | | | | | | 7 | 8ビット転送 (ロード) D←ソース | |
| LD D, A | 57 | | | | | | | } 4 | | |
| LD D, B | 50 | | | | | | | | | |
| LD D, C | 51 | | | | | | | | | |
| LD D, D | 52 | | | | | | | | | |
| LD D, E | 53 | - | - | - | - | - | - | | | |
| LD D, H | 54 | | | | | | | } 4 | | |
| LD D, L | 55 | | | | | | | | | |
| LD D, (HL) | 56 | | | | | | | | | |
| LD D, (IX+d) | DD 56 <u>d</u> | | | | | | | 7 | | |
| | | | | | | | | 19 | | |
| LD D, (IY+d) | FD 56 <u>d</u> | | | | | | | 19 | | |
| | | | | | | | | | | |
| LD E, n | 1E <u>n</u> | | | | | | | 7 | 8ビット転送 (ロード) E←ソース | |
| LD E, A | 5F | | | | | | | } 4 | | |
| LD E, B | 58 | | | | | | | | | |
| LD E, C | 59 | | | | | | | | | |
| LD E, D | 5A | | | | | | | | | |
| LD E, E | 5B | - | - | - | - | - | - | | | |
| LD E, H | 5C | | | | | | | } 4 | | |
| LD E, L | 5D | | | | | | | | | |
| LD E, (HL) | 5E | | | | | | | | | |
| LD E, (IX+d) | DD 5E <u>d</u> | | | | | | | 7 | | |
| | | | | | | | | 19 | | |
| LD E, (IY+d) | FD 5E <u>d</u> | | | | | | | 19 | | |
| | | | | | | | | | | |
| LD H, n | 26 <u>n</u> | | | | | | | 7 | 8ビット転送 (ロード) H←ソース | |
| LD H, A | 67 | | | | | | | } | | |
| LD H, B | 60 | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|--------------|-----------------------|-------|---|---|-----|---|---|---|---------------------|------------------------------------|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| LD H, C | 61 | | | | | | | | } 4 | |
| LD H, D | 62 | | | | | | | | | |
| LD H, E | 63 | - | - | - | - | - | - | - | | |
| LD H, H | 64 | | | | | | | | | |
| LD H, L | 65 | | | | | | | | | |
| LD H, (HL) | 66 | | | | | | | | 7 | |
| LD H, (IX+d) | DD 66 <u>d</u> | | | | | | | | 19 | |
| LD H, (IY+d) | FD 66 <u>d</u> | | | | | | | | 19 | |
| LD L, n | 2E <u>n</u> | | | | | | | | 7 | 8ビット転送 (ロード) L←ソース |
| LD L, A | 6F | | | | | | | | } 4 | |
| LD L, B | 68 | | | | | | | | | |
| LD L, C | 69 | | | | | | | | | |
| LD L, D | 6A | | | | | | | | | |
| LD L, E | 6B | - | - | - | - | - | - | - | | |
| LD L, H | 6C | | | | | | | | } 7 | |
| LD L, L | 6D | | | | | | | | | |
| LD L, (HL) | 6E | | | | | | | | | |
| LD L, (IX+d) | DD 6E <u>d</u> | | | | | | | | 19 | |
| LD L, (IY+d) | FD 6E <u>d</u> | | | | | | | | 19 | |
| LD I, A | ED 47 | - | - | - | - | - | - | - | 9 | 8ビット転送 I←A R←A |
| LD R, A | ED 4F | - | - | - | - | - | - | - | 9 | |
| LD (nn'), A | 32 <u>n'</u> <u>n</u> | | | | | | | | 13 | 8ビット転送 (nn')←A (BC)←A (DE)←A |
| LD (BC), A | 02 | - | - | - | - | - | - | - | 7 | |
| LD (DE), A | 12 | | | | | | | | 7 | |
| LD (HL), n | 36 <u>n</u> | | | | | | | | 10 | 8ビット転送 (ロード) (HL)←ソース |
| LD (HL), A | 77 | | | | | | | | } 7 | |
| LD (HL), B | 70 | | | | | | | | | |
| LD (HL), C | 71 | | | | | | | | | |
| LD (HL), D | 72 | - | - | - | - | - | - | - | | |
| LD (HL), E | 73 | | | | | | | | | |
| LD (HL), H | 74 | | | | | | | | | |
| LD (HL), L | 75 | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | |
|--|---|-------|---|---|-----|---|---|---|---------------------------|----------------------------------|---|
| | | S | Z | H | P/V | | N | C | | | |
| | | | | | P | V | | | | | |
| LD (IX+d),n LD (IX+d),A LD (IX+d),B LD (IX+d),C LD (IX+d),D LD (IX+d),E LD (IX+d),H LD (IX+d),L | DD 36 <u>d</u> , <u>n</u> DD 77 <u>d</u> DD 70 <u>d</u> DD 71 <u>d</u> DD 72 <u>d</u> DD 73 <u>d</u> DD 74 <u>d</u> DD 75 <u>d</u> | | | | | | | | 19 | 8ビット転送 (ロード) (IX+d) ← ソース | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| LD (IY+d),n LD (IY+d),A LD (IY+d),B LD (IY+d),C LD (IY+d),D LD (IY+d),E LD (IY+d),H LD (IY+d),L | FD 36 <u>d</u> , <u>n</u> FD 77 <u>d</u> FD 70 <u>d</u> FD 71 <u>d</u> FD 72 <u>d</u> FD 73 <u>d</u> FD 74 <u>d</u> FD 75 <u>d</u> | | | | | | | | 19 | 8ビット転送 (ロード) (IY+d) ← ソース | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| LD BC, nn' LD BC, (nn') | 01 <u>n'</u> , <u>n</u> ED 4B <u>n'</u> , <u>n</u> | | | | | | | | 10 20 | 16ビット転送 BC ← ソース | nn' : 定数 (nn') : メモリの内容 メモリからレジスタ の場合, たとえば HL, (nn') では L ← (nn') H ← (nn'+1) となる |
| LD DE, nn' LD DE, (nn') | 11 <u>n'</u> , <u>n</u> ED 5B <u>n'</u> , <u>n</u> | | | | | | | | 10 20 | 16ビット転送 DE ← ソース | |
| LD HL, nn' LD HL, (nn') | 21 <u>n'</u> , <u>n</u> 2A <u>n'</u> , <u>n</u> | | | | | | | | 10 16 | 16ビット転送 HL ← ソース | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| LD SP, nn' LD SP, (nn') LD SP, HL LD SP, IX LD SP, IY | 31 <u>n'</u> , <u>n</u> ED 7B <u>n'</u> , <u>n</u> F9 DD F9 FD F9 | | | | | | | | 10 20 6 10 10 | 16ビット転送 SP ← ソース | |
| LD IX, nn' LD IX, (nn') | DD 21 <u>n'</u> , <u>n</u> DD 2A <u>n'</u> , <u>n</u> | | | | | | | | 14 20 | 16ビット転送 IX ← ソース | |

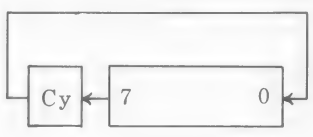
| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|--------------|------------------|-------|---|---|-----|---|---|---|----------------------------------|---|
| | | S | Z | H | P/V | | N | C | | |
| LD IY, nn' | FD 21 <u>nn'</u> | — | — | — | — | — | — | — | 14 | 16 ビット転送 IY ← ソース |
| LD IY, (nn') | FD 2A <u>nn'</u> | — | — | — | — | — | — | — | 20 | |
| LD (nn'), BC | ED 43 <u>nn'</u> | | | | | | | | 20 | 16 ビット転送 (ロード) (nn') ← ソース L (nn' + 1) ← ソース H |
| LD (nn'), DC | ED 53 <u>nn'</u> | | | | | | | | 20 | |
| LD (nn'), HL | 22 <u>nn'</u> | — | — | — | — | — | — | — | 16 | |
| LD (nn'), SP | ED 73 <u>nn'</u> | | | | | | | | 20 | |
| LD (nn'), IX | DD 22 <u>nn'</u> | | | | | | | | 20 | |
| LD (nn'), IY | FD 22 <u>nn'</u> | | | | | | | | 20 | |
| LDD | ED A8 | × | × | 0 | — | • | 0 | — | 16 | ブロック転送 (ロード・ディクリメント) (DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1 |
| LDDR | ED B8 | × | × | 0 | — | 0 | 0 | — | 1 バイト につき 21 最終のみ 16 | ブロック転送 (ロード・ディクリメント・ リピート) LDDを BC = 0 までくり返す |
| LDI | ED A0 | × | × | 0 | — | • | 0 | — | 16 | ブロック転送 (ロード・インクリメント) (DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1 |
| LDIR | ED B0 | × | × | 0 | — | 0 | 0 | — | 1 バイト につき 21 最終のみ 16 | ブロック転送 (ロード・インクリメント・ リピート) LDIを BC = 0 までくり返す |
| NEG | ED 44 | • | • | • | — | • | 1 | • | 8 | ニゲイト 2の補数をとる A ← 0 - A |
| NOP | 00 | — | — | — | — | — | — | — | 4 | 何もしないで次へ (ノーオペレーション) |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 | | | | | | | | | | | | | | | | |
|------------|----------------|-------|---|---|-----|---|---|---------------------|-----------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S | Z | H | P/V | | N | | | C | | | | | | | | | | | | | | | |
| | | | | | P | V | | | | | | | | | | | | | | | | | | | |
| OR n | F6 <u>n</u> | | | | | | | | 7 | 論理和 (オア) A ← A ∨ ソース <table><tr><th>a</th><th>b</th><th>答</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | a | b | 答 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| a | b | 答 | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| OR A | B7 | | | | | | | | 4 | | | | | | | | | | | | | | | | |
| OR B | B0 | | | | | | | | | | | | | | | | | | | | | | | | |
| OR C | B1 | | | | | | | | | | | | | | | | | | | | | | | | |
| OR D | B2 | | | | | | | | | | | | | | | | | | | | | | | | |
| OR E | B3 | • | • | 0 | • | — | 0 | 0 | | | | | | | | | | | | | | | | | |
| OR H | B4 | | | | | | | | | | | | | | | | | | | | | | | | |
| OR L | B5 | | | | | | | | | | | | | | | | | | | | | | | | |
| OR (HL) | B6 | | | | | | | | 7 | | | | | | | | | | | | | | | | |
| OR (IX+d) | DD B6 <u>d</u> | | | | | | | | 19 | | | | | | | | | | | | | | | | |
| OR (IY+d) | FD B6 <u>d</u> | | | | | | | | 19 | | | | | | | | | | | | | | | | |
| OUT (C), A | ED 79 | | | | | | | | 12 | 出力 各レジスタの内容を BC レジスタの内容番地のポートへ出力 〔 I/O (BC) ← ソース 〕 | | | | | | | | | | | | | | | |
| OUT (C), B | ED 41 | | | | | | | | | | | | | | | | | | | | | | | | |
| OUT (C), C | ED 49 | | | | | | | | | | | | | | | | | | | | | | | | |
| OUT (C), D | ED 51 | — | — | — | — | — | — | — | | | | | | | | | | | | | | | | | |
| OUT (C), E | ED 59 | | | | | | | | | | | | | | | | | | | | | | | | |
| OUT (C), H | ED 61 | | | | | | | | | | | | | | | | | | | | | | | | |
| OUT (C), L | ED 69 | | | | | | | | | | | | | | | | | | | | | | | | |
| OUT (n), A | D3 <u>n</u> | — | — | — | — | — | — | — | 11 | 出力 I/O (An) ← A | | | | | | | | | | | | | | | |
| OUTD | ED AB | × | • | × | × | — | × | × | 16 | アウト・ディクリメント I/O(BC)←(HL) HL←HL-1 B←B-1 | | | | | | | | | | | | | | | |
| OTDR | ED BB | × | 1 | × | × | — | × | × | 1バイトにつき 21 最終のみ 16 | アウト・ディクリメント・リピート OUTDをB=0までくり返す | | | | | | | | | | | | | | | |
| OUTI | ED A3 | × | • | × | × | — | × | × | 16 | アウト・インクリメント I/O(BC)←(HL) HL←HL+1 B←B-1 | | | | | | | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | | |
|---------------|-------------------|-------|---|---|-----|---|---|---|---------------------|-----|-----------------------------|---|
| | | S | Z | H | P/V | | N | C | | | | |
| | | | | | P | V | | | | | | |
| OTIR | ED B3 | | × | 1 | × | × | | — | × | × | 1バイトにつき 21 最終のみ 16 | アウト・インクリメント・リピート OUTIをB=0までくり返す |
| POP AF | F1 | | | | | | | | | | } 10 14 14 | 16ビット転送 (ポップ) スタックからレジスタへ転送 ディスティネーションL←(SP) ディスティネーションH← (SP+1) SP←SP+2 |
| POP BC | C1 | | | | | | | | | | | |
| POP DE | D1 | | | | | | | | | | | |
| POP HL | E1 | — | — | — | — | — | — | — | — | — | | |
| POP IX | DD E1 | | | | | | | | | | | |
| POP IY | FD E1 | | | | | | | | | | | |
| PUSH AF | F5 | | | | | | | | | | } 11 15 15 | 16ビット転送 (プッシュ) レジスタからスタックへ転送 (SP-1)←ソースH (SP-2)←ソースL SP←SP-2 |
| PUSH BC | C5 | | | | | | | | | | | |
| PUSH DE | D5 | — | — | — | — | — | — | — | — | — | | |
| PUSH HL | E5 | | | | | | | | | | | |
| PUSH IX | DD E5 | | | | | | | | | | | |
| PUSH IY | FD E5 | | | | | | | | | | | |
| RES O, A | CB 87 | | | | | | | | | | } 8 15 23 23 | ビットリセット ソースの第0ビット←0 |
| RES O, B | CB 80 | | | | | | | | | | | |
| RES O, C | CB 81 | | | | | | | | | | | |
| RES O, D | CB 82 | | | | | | | | | | | |
| RES O, E | CB 83 | — | — | — | — | — | — | — | — | — | | |
| RES O, H | CB 84 | | | | | | | | | | | |
| RES O, L | CB 85 | | | | | | | | | | | |
| RES O, (HL) | CB 86 | | | | | | | | | | | |
| RES O, (IX+d) | DD CB <u>d</u> 86 | | | | | | | | | | | |
| RES O, (IY+d) | FD CB <u>d</u> 86 | | | | | | | | | | | |
| RES 1, A | CB 8F | | | | | | | | | | } 8 15 | ビットリセット ソースの第1ビット←0 |
| RES 1, B | CB 88 | | | | | | | | | | | |
| RES 1, C | CB 89 | | | | | | | | | | | |
| RES 1, D | CB 8A | | | | | | | | | | | |
| RES 1, E | CB 8B | — | — | — | — | — | — | — | — | — | | |
| RES 1, H | CB 8C | | | | | | | | | | | |
| RES 1, L | CB 8D | | | | | | | | | | | |
| RES 1, (HL) | CB 8E | | | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 |
|---------------|-------------------|-------|---|---|------------|---|---|------------------------|----------------------------|
| | | S | Z | H | P/V P V | N | C | | |
| RES 1, (IX+d) | DD CB <u>d</u> 8E | | | | | | | 23 | |
| RES 1, (IY+d) | FD CB <u>d</u> 8E | | | | | | | 23 | |
| RES 2, A | CB 97 | | | | | | | } 8 | ビットリセット ソースの第2ビット←0 |
| RES 2, B | CB 90 | | | | | | | | |
| RES 2, C | CB 91 | | | | | | | | |
| RES 2, D | CB 92 | | | | | | | | |
| RES 2, E | CB 93 | | | | | | | | |
| RES 2, H | CB 94 | - | - | - | - | - | - | | |
| RES 2, L | CB 95 | | | | | | | | |
| RES 2, (HL) | CB 96 | | | | | | | 15 | |
| RES 2, (IX+d) | DD CB <u>d</u> 96 | | | | | | | 23 | |
| RES 2, (IY+d) | FD CB <u>d</u> 96 | | | | | | | 23 | |
| RES 3, A | CB 9F | | | | | | | } 8 | ビットリセット ソースの第3ビット←0 |
| RES 3, B | CB 98 | | | | | | | | |
| RES 3, C | CB 99 | | | | | | | | |
| RES 3, D | CB 9A | | | | | | | | |
| RES 3, E | CB 9B | | | | | | | | |
| RES 3, H | CB 9C | - | - | - | - | - | - | | |
| RES 3, L | CB 9D | | | | | | | | |
| RES 3, (HL) | CB 9E | | | | | | | 15 | |
| RES 3, (IX+d) | DD CB <u>d</u> 9E | | | | | | | 23 | |
| RES 3, (IY+d) | FD CB <u>d</u> 9E | | | | | | | 23 | |
| RES 4, A | CB A7 | | | | | | | } 8 | ビットリセット ソースの第4ビット←0 |
| RES 4, B | CB A0 | | | | | | | | |
| RES 4, C | CB A1 | | | | | | | | |
| RES 4, D | CB A2 | | | | | | | | |
| RES 4, E | CB A3 | | | | | | | | |
| RES 4, H | CB A4 | - | - | - | - | - | - | | |
| RES 4, L | CB A5 | | | | | | | | |
| RES 4, (HL) | CB A6 | | | | | | | 15 | |
| RES 4, (IX+d) | DD CB <u>d</u> A6 | | | | | | | 23 | |
| RES 4, (IY+d) | FD CB <u>d</u> A6 | | | | | | | 23 | |
| RES 5, A | CB AF | | | | | | | } | ビットリセット |
| RES 5, B | CB A8 | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 | |
|---------------|-------------------|-------|---|---|-----|---|---|---------------------|--------------------------------|----|
| | | S | Z | H | P/V | | N | | | C |
| | | | | | P | V | | | | |
| RES 5, C | CB A9 | | | | | | | } 8 | ソースの第5ビット← 0 | |
| RES 5, D | CB AA | | | | | | | | | |
| RES 5, E | CB AB | | | | | | | | | |
| RES 5, H | CB AC | - | - | - | - | - | - | | | |
| RES 5, L | CB AD | | | | | | | | | |
| RES 5, (HL) | CB AE | | | | | | | 15 | | |
| RES 5, (IX+d) | DD CB <u>d</u> AE | | | | | | | 23 | | |
| RES 5, (IY+d) | FD CB <u>d</u> AE | | | | | | | 23 | | |
| RES 6, A | CB B7 | | | | | | | } 8 | ビットリセット ソースの第6ビット← 0 | |
| RES 6, B | CB B0 | | | | | | | | | |
| RES 6, C | CB B1 | | | | | | | | | |
| RES 6, D | CB B2 | | | | | | | | | |
| RES 6, E | CB B3 | | | | | | | | | |
| RES 6, H | CB B4 | - | - | - | - | - | - | 15 | | |
| RES 6, L | CB B5 | | | | | | | | | |
| RES 6, (HL) | CB B6 | | | | | | | | | |
| RES 6, (IX+d) | DD CB <u>d</u> B6 | | | | | | | 23 | | |
| RES 6, (IY+d) | FD CB <u>d</u> B6 | | | | | | | 23 | | |
| RES 7, A | CB BF | | | | | | | } 8 | ビットリセット ソースの第7ビット← 0 | |
| RES 7, B | CB B8 | | | | | | | | | |
| RES 7, C | CB B9 | | | | | | | | | |
| RES 7, D | CB BA | | | | | | | | | |
| RES 7, E | CB BB | - | - | - | - | - | - | | | 15 |
| RES 7, H | CB BC | | | | | | | | | |
| RES 7, L | CB BD | | | | | | | | | |
| RES 7, (HL) | CB BE | | | | | | | 23 | | |
| RES 7, (IX+d) | DD CB <u>d</u> BE | | | | | | | 23 | | |
| RES 7, (IY+d) | FD CB <u>d</u> BE | | | | | | | 23 | | |
| RET | C9 | - | - | - | - | - | - | 10 | サブルーチンからのリターン PCヘスタックよりPOP | |
| RET NZ | C0 | | | | | | | 条件成立 11 | 条件付リターン ・条件が成立すれば POP PC | |
| RET Z | C8 | | | | | | | | | |
| RET NC | D0 | | | | | | | | | |

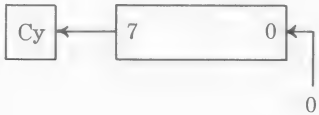
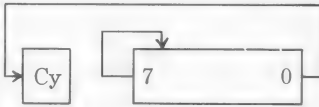
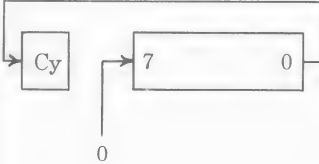
| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|-----------|-------------------|-------|---|---|-----|---|---|---|------------------------|---|
| | | S | Z | H | P/V | | N | C | | |
| RET C | D8 | — | — | — | — | — | — | — | 不 成 立 5 | ・ 不成立なら本命令は無視する |
| RET PO | E0 | | | | | | | | | |
| RET PE | E8 | | | | | | | | | |
| RET P | F0 | | | | | | | | | |
| RET M | F8 | | | | | | | | | |
| RETI | ED 4D | — | — | — | — | — | — | — | 14 | 割り込み処理からのリターン POP PC |
| RETN | ED 45 | — | — | — | — | — | — | — | 14 | ノンマスカブル割り込み処理からのリターン POP PC |
| RLA | 17 | — | — | 0 | — | — | 0 | ・ | 4 | ローテート・レフト・アキュムレーター RL Aと同じ |
| RLCA | 07 | — | — | 0 | — | — | 0 | ・ | 4 | ローテート・レフト・サーキュラ・アキュムレーター RLC Aと同じ |
| RRA | 1F | — | — | 0 | — | — | 0 | ・ | 4 | ローテート・ライト・アキュムレーター RR Aと同じ |
| RRCA | 0F | — | — | 0 | — | — | 0 | ・ | 4 | ローテート・ライト・サーキュラ・アキュムレーター RRC Aと同じ |
| RL A | CB 17 | | | | | | | | } 8 | ローテート・レフト  |
| RL B | CB 10 | | | | | | | | | |
| RL C | CB 11 | | | | | | | | | |
| RL D | CB 12 | | | | | | | | | |
| RL E | CB 13 | | | | | | | | | |
| RL H | CB 14 | ・ | ・ | 0 | ・ | — | 0 | ・ | | |
| RL L | CB 15 | | | | | | | | | |
| RL (HL) | CB 16 | | | | | | | | | |
| RL (IX+d) | DD CB <u>d</u> 16 | | | | | | | | 15 | |
| RL (IY+d) | FF CB <u>d</u> 16 | | | | | | | | 23 | |
| | | | | | | | | | 23 | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 | |
|------------|-------------------|-------|---|---|-----|---|---|---------------------|----------------------|---|
| | | S | Z | H | P/V | | N | | | C |
| | | | | | P | V | | | | |
| RLC A | CB 07 | | | | | | | 8 | ローテート・レフト・サーキュラ | |
| RLC B | CB 00 | | | | | | | | | |
| RLC C | CB 01 | | | | | | | | | |
| RLC D | CB 02 | | | | | | | | | |
| RLC E | CB 03 | • | • | 0 | • | - | 0 | | | • |
| RLC H | CB 04 | | | | | | | | | |
| RLC L | CB 05 | | | | | | | | | |
| RLC (HL) | CB 06 | | | | | | | 15 | | |
| RLC (IX+d) | DD CB <u>d</u> 06 | | | | | | | 23 | | |
| RLC (IY+d) | FD CB <u>d</u> 06 | | | | | | | 23 | | |
| RR A | CB 1F | | | | | | | 8 | ローテート・ライト | |
| RR B | CB 18 | | | | | | | | | |
| RR C | CB 19 | | | | | | | | | |
| RR D | CB 1A | | | | | | | | | |
| RR E | CB 1B | • | • | 0 | • | - | 0 | | | • |
| RR H | CB 1C | | | | | | | | | |
| RR L | CB 1D | | | | | | | | | |
| RR (HL) | CB 1E | | | | | | | 15 | | |
| RR (IX+d) | DD CB <u>d</u> 1E | | | | | | | 23 | | |
| RR (IY+d) | FD CB <u>d</u> 1E | | | | | | | 23 | | |
| RRC A | CB 0F | | | | | | | 8 | ローテート・ライト・サーキュラ | |
| RRC B | CB 08 | | | | | | | | | |
| RRC C | CB 09 | | | | | | | | | |
| RRC D | CB 0A | | | | | | | | | |
| RRC E | CB 0B | • | • | 0 | • | - | 0 | | | • |
| RRC H | CB 0C | | | | | | | | | |
| RRC L | CB 0D | | | | | | | | | |
| RRC (HL) | CB 0E | | | | | | | 15 | | |
| RRC (IX+d) | DD CB <u>d</u> 0E | | | | | | | 23 | | |
| RRC (IY+d) | FD CB <u>d</u> 0E | | | | | | | 23 | | |
| RLD | ED 6F | | | | | | | 18 | ローテート・レフト・ディジット A | |
| | | • | • | 0 | • | - | 0 | - | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | |
|---|---|-------|---|---|-----|---|---|---|------------------------|------------------------------|--|
| | | S | Z | H | P/V | | N | C | | | |
| | | | | | P | V | | | | | |
| RRD | ED 67 | | | | | | | | | 18 | ローテート・ライト・ディジット <div></div> |
| RST 00H RST 08H RST 10H RST 18H RST 20H RST 28H RST 30H RST 38H | C7 CF D7 DF E7 EF F7 FF | | | | | | | | | 11 | リスタート 0000 H ~ 0038 Hのいずれか の番地に対する CALL |
| SBC A, n SBC A, A SBC A, B SBC A, C SBC A, D SBC A, E SBC A, H SBC A, L SBC A, (HL) SBC A, (IX+d) SBC A, (IY+d) | DE <u>n</u> 9F 98 99 9A 9B 9C 9D 9E DD 9E <u>d</u> FD 9E <u>d</u> | | | | | | | | | 7 4 4 7 19 19 | 8ビット引き算 (キャリ付) (サブトラクト・ウィズ・キャリ) A←A-ソース-Cy |
| SBC HL, BC SBC HL, DE SBC HL, HL SBC HL, SP | ED 42 ED 52 ED 62 ED 72 | | | | | | | | | 15 | 16ビット引き算 (キャリ付) (サブトラクト・ウィズ・キャリ) HL←HL-ソース-Cy |
| SCF | 37 | | | | | | | | | 4 | セット・キャリフラグ Cy←1 |
| SET 0, A SET 0, B SET 0, C | CB C7 CB C0 CB C1 | | | | | | | | | | ビットセット |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | |
|---------------|-------------------|-------|---|---|-----|---|---|---|------------------------|---------------------------|--|
| | | S | Z | H | P/V | | N | C | | | |
| | | | | | P | V | | | | | |
| SET 0, D | CB C2 | | | | | | | | } 8 | ソースの第0ビット←1 | |
| SET 0, E | CB C3 | - | - | - | - | - | - | - | | | |
| SET 0, H | CB C4 | | | | | | | | | | |
| SET 0, L | CB C5 | | | | | | | | | | |
| SET 0, (HL) | CB C6 | | | | | | | | 15 | | |
| SET 0, (IX+d) | DD CB <u>d</u> C6 | | | | | | | | 23 | | |
| SET 0, (IY+d) | FD CB <u>d</u> C6 | | | | | | | | 23 | | |
| SET 1, A | CB CF | | | | | | | | } 8 | ビットセット ソースの第1ビット←1 | |
| SET 1, B | CB C8 | | | | | | | | | | |
| SET 1, C | CB C9 | | | | | | | | | | |
| SET 1, D | CB CA | | | | | | | | | | |
| SET 1, E | CB CB | - | - | - | - | - | - | - | } 15 | | |
| SET 1, H | CB CC | | | | | | | | | | |
| SET 1, L | CB CD | | | | | | | | | | |
| SET 1, (HL) | CB CE | | | | | | | | 15 | | |
| SET 1, (IX+d) | DD CB <u>d</u> CE | | | | | | | | 23 | | |
| SET 1, (IY+d) | FD CB <u>d</u> CE | | | | | | | | 23 | | |
| SET 2, A | CB D7 | | | | | | | | } 8 | ビットセット ソースの第2ビット←1 | |
| SET 2, B | CB D0 | | | | | | | | | | |
| SET 2, C | CB D1 | | | | | | | | | | |
| SET 2, D | CB D2 | | | | | | | | | | |
| SET 2, E | CB D3 | - | - | - | - | - | - | - | } 15 | | |
| SET 2, H | CB D4 | | | | | | | | | | |
| SET 2, L | CB D5 | | | | | | | | | | |
| SET 2, (HL) | CB D6 | | | | | | | | 15 | | |
| SET 2, (IX+d) | DD CB <u>d</u> D6 | | | | | | | | 23 | | |
| SET 2, (IY+d) | FD CB <u>d</u> D6 | | | | | | | | 23 | | |
| SET 3, A | CB DF | | | | | | | | } 8 | ビットセット ソースの第3ビット←1 | |
| SET 3, B | CB D8 | | | | | | | | | | |
| SET 3, C | CB D9 | | | | | | | | | | |
| SET 3, D | CB DA | | | | | | | | | | |
| SET 3, E | CB DB | - | - | - | - | - | - | - | } 15 | | |
| SET 3, H | CB DC | | | | | | | | | | |
| SET 3, L | CB DD | | | | | | | | | | |
| SET 3, (HL) | CB DE | | | | | | | | 15 | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 |
|---------------|-----------------------|-------|---|---|-----|---|---|---|------------------------|---------------------------|
| | | S | Z | H | P/V | | N | C | | |
| | | | | | P | V | | | | |
| SET 3, (IX+d) | DD CB _d DE | | | | | | | | 23 | |
| SET 3, (IY+d) | FD CB _d DE | | | | | | | | 23 | |
| SET 4, A | CB E7 | | | | | | | | } 8 | ビットセット ソースの第4ビット←1 |
| SET 4, B | CB E0 | | | | | | | | | |
| SET 4, C | CB E1 | | | | | | | | | |
| SET 4, D | CB E2 | | | | | | | | | |
| SET 4, E | CB E3 | | | | | | | | | |
| SET 4, H | CB E4 | - | - | - | - | - | - | | | |
| SET 4, L | CB E5 | | | | | | | | | |
| SET 4, (HL) | CB E6 | | | | | | | | 15 | |
| SET 4, (IX+d) | DD CB _d E6 | | | | | | | | 23 | |
| SET 4, (IY+d) | FD CB _d E6 | | | | | | | | 23 | |
| SET 5, A | CB EF | | | | | | | | } 8 | ビットセット ソースの第5ビット←1 |
| SET 5, B | CB E8 | | | | | | | | | |
| SET 5, C | CB E9 | | | | | | | | | |
| SET 5, D | CB EA | | | | | | | | | |
| SET 5, E | CB EB | | | | | | | | | |
| SET 5, H | CB EC | - | - | - | - | - | - | | | |
| SET 5, L | CB ED | | | | | | | | | |
| SET 5, (HL) | CB EE | | | | | | | | 15 | |
| SET 5, (IX+d) | DD CB _d EE | | | | | | | | 23 | |
| SET 5, (IY+d) | FD CB _d EE | | | | | | | | 23 | |
| SET 6, A | CB F7 | | | | | | | | } 8 | ビットセット ソースの第6ビット←1 |
| SET 6, B | CB F0 | | | | | | | | | |
| SET 6, C | CB F1 | | | | | | | | | |
| SET 6, D | CB F2 | | | | | | | | | |
| SET 6, E | CB F3 | | | | | | | | | |
| SET 6, H | CB F4 | - | - | - | - | - | - | | | |
| SET 6, L | CB F5 | | | | | | | | | |
| SET 6, (HL) | CB F6 | | | | | | | | 15 | |
| SET 6, (IX+d) | DD CB _d F6 | | | | | | | | 23 | |
| SET 6, (IY+d) | FD CB _d F6 | | | | | | | | 23 | |
| SET 7, A | CB FF | | | | | | | | } | ビットセット |
| SET 7, B | CB F8 | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | | 所 要 クロック サイクル | 動 作 | |
|---------------|-----------------------|-------|---|---|-----|---|---|---|------------------------|---|--|
| | | S | Z | H | P/V | | N | C | | | |
| | | | | | P | V | | | | | |
| SET 7, C | CB F9 | | | | | | | | 8 | ソースの第7ビット←1 | |
| SET 7, D | CB FA | | | | | | | | | | |
| SET 7, E | CB FB | | | | | | | | | | |
| SET 7, H | CB FC | — | — | — | — | — | — | — | | | |
| SET 7, L | CB FD | | | | | | | | 15 | | |
| SET 7, (HL) | CB FE | | | | | | | | | | |
| SET 7, (IX+d) | DD CB _d FE | | | | | | | | 23 | | |
| SET 7, (IY+d) | FD CB _d FE | | | | | | | | 23 | | |
| SLA A | CB 27 | | | | | | | | 8 | シフト・レフト・アリスメチック  | |
| SLA B | CB 20 | | | | | | | | | | |
| SLA C | CB 21 | | | | | | | | | | |
| SLA D | CB 22 | | | | | | | | | | |
| SLA E | CB 23 | • | • | 0 | • | — | 0 | • | 15 | | |
| SLA H | CB 24 | | | | | | | | | | |
| SLA L | CB 25 | | | | | | | | | | |
| SLA (HL) | CB 26 | | | | | | | | | | |
| SLA (IX+d) | DD CB _d 26 | | | | | | | | 23 | | |
| SLA (IY+d) | FD CB _d 26 | | | | | | | | 23 | | |
| SRA A | CB 2F | | | | | | | | 8 | シフト・ライト・アリスメチック  | |
| SRA B | CB 28 | | | | | | | | | | |
| SRA C | CB 29 | | | | | | | | | | |
| SRA D | CB 2A | | | | | | | | | | |
| SRA E | CB 2B | • | • | 0 | • | — | 0 | • | 15 | | |
| SRA H | CB 2C | | | | | | | | | | |
| SRA L | CB 2D | | | | | | | | | | |
| SRA (HL) | CB 2E | | | | | | | | | | |
| SRA (IX+d) | DD CB _d 2E | | | | | | | | 23 | | |
| SRA (IY+d) | FD CB _d 2E | | | | | | | | 23 | | |
| SRL A | CB 3F | | | | | | | | 8 | シフト・ライト・ロジカル  | |
| SRL B | CB 38 | | | | | | | | | | |
| SRL C | CB 39 | | | | | | | | | | |
| SRL D | CB 3A | | | | | | | | | | |
| SRL E | CB 3B | • | • | 0 | • | — | 0 | • | 15 | | |
| SRL H | CB 3C | | | | | | | | | | |
| SRL L | CB 3D | | | | | | | | | | |

| ニーモニック | マシン語 | フラグ変化 | | | | | | 所 要 クロック サイクル | 動 作 | | | | | | | | | | | | | | | | |
|------------|-----------|-------|---|---|-----|---|---|---------------------|-----|------------------------------------|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S | Z | H | P/V | | N | | | C | | | | | | | | | | | | | | | |
| | | | | | P | V | | | | | | | | | | | | | | | | | | | |
| SRL (HL) | CB 3E | | | | | | | | 15 | | | | | | | | | | | | | | | | |
| SRL (IX+d) | DD CB,d3E | | | | | | | | 23 | | | | | | | | | | | | | | | | |
| SRL (IY+d) | FD CB,d3E | | | | | | | | 23 | | | | | | | | | | | | | | | | |
| SUB n | D6 n | | | | | | | | 7 | 8ビット引き算 (サブトラクト) A←A－ソース | | | | | | | | | | | | | | | |
| SUB A | 97 | | | | | | | | 4 | | | | | | | | | | | | | | | | |
| SUB B | 90 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB C | 91 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB D | 92 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB E | 93 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB H | 94 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB L | 95 | | | | | | | | | | | | | | | | | | | | | | | | |
| SUB (HL) | 96 | | | | | | | 7 | | | | | | | | | | | | | | | | | |
| SUB (IX+d) | DD 96 d | | | | | | | 19 | | | | | | | | | | | | | | | | | |
| SUB (IY+d) | FD 96 d | | | | | | | 19 | | | | | | | | | | | | | | | | | |
| XOR n | EE n | | | | | | | | 7 | | 排他的論理和 (エクスクルーシブ・オア) A←A ⊕ ソース <table><tr><td>a</td><td>b</td><td>答</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | a | b | 答 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| a | b | 答 | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | |
| XOR A | AF | | | | | | | | 4 | | | | | | | | | | | | | | | | |
| XOR B | A8 | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR C | A9 | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR D | AA | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR E | AB | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR H | AC | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR L | AD | | | | | | | | | | | | | | | | | | | | | | | | |
| XOR (HL) | AE | | | | | | | 7 | | | | | | | | | | | | | | | | | |
| XOR (IX+d) | DD AE d | | | | | | | 19 | | | | | | | | | | | | | | | | | |
| XOR (IY+d) | FD AE d | | | | | | | 19 | | | | | | | | | | | | | | | | | |

| a | b | 答 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

付録2 1 バイト符号付 16 進数

(欄外は、16 進数)
(欄内は、10 進数)

| 上位 \ 下位 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 2 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 4 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 5 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 6 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 7 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 8 | -128 | -127 | -126 | -125 | -124 | -123 | -122 | -121 | -120 | -119 | -118 | -117 | -116 | -115 | -114 | -113 |
| 9 | -112 | -111 | -110 | -109 | -108 | -107 | -106 | -105 | -104 | -103 | -102 | -101 | -100 | -99 | -98 | -97 |
| A | -96 | -95 | -94 | -93 | -92 | -91 | -90 | -89 | -88 | -87 | -86 | -85 | -84 | -83 | -82 | -81 |
| B | -80 | -79 | -78 | -77 | -76 | -75 | -74 | -73 | -72 | -71 | -70 | -69 | -68 | -67 | -66 | -65 |
| C | -64 | -63 | -62 | -61 | -60 | -59 | -58 | -57 | -56 | -55 | -54 | -53 | -52 | -51 | -50 | -49 |
| D | -48 | -47 | -46 | -45 | -44 | -43 | -42 | -41 | -40 | -39 | -38 | -37 | -36 | -35 | -34 | -33 |
| E | -32 | -31 | -30 | -29 | -28 | -27 | -26 | -25 | -24 | -23 | -22 | -21 | -20 | -19 | -18 | -17 |
| F | -16 | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

付録3 チェック・サム・プログラム

```

10 /
20 /
30 / チェック・サム プログラム
40 /
50 / by Yasuhiro Shimizu
60 /
70 / 1983.10.15
80 /
90 /
100 /
110 /
120 / ショキカ
130 /
140 CLEAR &HCFF0 :OPTION BASE 0 :DEFINT M,C,D :DIM M(15,7),C(15),D(7)
150 INIT :WIDTH 40 :CLS 4 :COLOR 4
160 /
170 / セツメイ
180 /
190 PRINT "XOKK チェック・サム プログラム XOKK" :PRINT :COLOR 3
200 PRINT "コノ プログラム ハ、シタイ サレタ ハンイ ノ"
210 PRINT "メモリ ノ ナイヨウ ヲ チェック・サム ヒョウシ"
220 PRINT "シマス。" :PRINT :PRINT :COLOR 6
230 PRINT "[チュウイ] セイカク ヲ キスタメニ、シタイハンイ イコウ ノ"
240 PRINT "メモリ ナイヨウ ヲ スベテ 0 ニ シマスノデ"
250 PRINT "ヒツヨウ ナラ、SAVE シタイテ クダサイ。" :PRINT :COLOR 2
260 PRINT "ヨウイ ハ イテスカ ? " :CFLASH 1 :COLOR 7
270 PRINT "YES = リターン キー、 NO = ソノタ ノ キー " :CFLASH 0
280 REPEAT :I$=INKEY$ :UNTIL I$<>" "
290 IF I$<>CHR$(13) THEN CLS :END
300 /
310 / マシンコト カキコエ
320 /
330 RESTORE 3000
340 AD=&HCFF0 :マシニング・エリア = &HCFF0 カラ &HCFFD マデ
350 FOR I=0 TO 13
360 READ MC$ :POKE AD+I,VAL("&H"+MC$)
370 NEXT
380 DEF USR=&HCFF0
390 /
400 / ニュウリョク
410 /
420 CLS :COLOR 7 :CFLASH 1
430 PRINT "ADDRESS ハ D000 イコウ ニ シタイシテ クダサイ !" :CFLASH 0 :COLOR 4 :PRINT :PRINT
440 BEEP :INPUT "START ADDRESS (16シン 4ケタ) = " :ST$ :ST=VAL("&H"+ST$) :IF ST<0 THEN ST=ST+65536!
450 IF ST<53248! THEN PRINT CHR$(7,30,5) :GOTO 440
460 COLOR 6
470 BEEP :INPUT " END ADDRESS (16シン 4ケタ) = " :ED$ :ED=VAL("&H"+ED$) :IF ED<0 THEN ED=ED+65536!
480 IF ED<ST THEN PRINT CHR$(7,30,5) :GOTO 470
490 PRINT :PRINT :COLOR 3 :BEEP
500 PRINT "タイセイ シマスカ ? [ NO=リターン・キー、YES=Y ] "
510 REPEAT :I$=INKEY$ :UNTIL I$<>" "
520 IF INSTR(1,"Yy",I$)>0 THEN 420
530 LOCATE 0,20 :COLOR 5 :CFLASH 1
540 PRINT "チェック・サム ケイサン チュウ" :CFLASH 0
550 /
560 / メモリー・クリア
570 /
580 REM ——> クリア ハ ED+1 カラ &HFEFF マデ (&HFF00 カラ ハ モニター ワーク・エリア)
590 /
600 AX=USR(CINT(ED+1))
610 /
620 / メイン ルーチン
630 /
640 AD=ST
650 /
660 / —— 128 ハイト ユニトリ ト ヨコ ノ SUM ——
670 /
680 FOR I=0 TO 15
690 C(I)=0
700 FOR J=0 TO 7
710 M(I,J)=PEEK(AD+I*8+J) :C(I)=C(I)+M(I,J)
720 NEXT
730 NEXT
740 /
750 / —— タテ ノ SUM トーグル ——
760 /
770 S=0
780 FOR J=0 TO 7
790 D(J)=0
800 FOR I=0 TO 15
810 D(J)=D(J)+M(I,J)

```

```

820 NEXT
830 S=S+D(J)
840 NEXT
850 /
860 / — チェック・サム ヒョウシ —
870 /
880 INIT :CLS :COLOR 4
890 PRINT "***** Check Sum *****" :PRINT :COLOR 7
900 FOR I=0 TO 15
910 PRINT " :"+RIGHT$("000"+HEX$(AD+I*8),4)+" = ";
920 FOR J=0 TO 7
930 PRINT RIGHT$("0"+HEX$(M(I,J)),2)+" ";
940 NEXT
950 PRINT " : "+RIGHT$("0"+HEX$(C(I)),2)
960 NEXT
970 PRINT
980 FOR I=1 TO 37 :PRINT "—"; :NEXT :PRINT
990 PRINT " ";
1000 FOR J=0 TO 7
1010 PRINT RIGHT$("0"+HEX$(D(J)),2)+" ";
1020 NEXT
1030 PRINT " : "+RIGHT$("0"+HEX$(S),2)
1040 /
1050 / — タイペイ —
1060 /
1070 CONSOLE 23,2 :CLS :COLOR 6
1080 BEEP :PRINT "タイペイ カシヨ ム アリマスカ ? [ NO=リターン・キー YES=Y ] ";
1090 REPEAT :I$=INKEY$ :UNTIL I$<>" "
1100 IF INSTR(1,"Yy"),I$)=0 THEN 1230
1110 ADR$="" :BEEP :COLOR 3 :PRINT :INPUT "タイペイ・アドレス ム [ 16シン 47タ ] ";ADR$
1120 ADR=VAL("&H"+ADR$) :IF ADR<0 THEN ADR=ADR+65536!
1130 IF ADR<AD OR ADR>(AD+127) THEN CLS :BEEP :GOTO 1110
1140 MC$=RIGHT$("0"+HEX$(PEEK(ADR)),2)
1150 MC1$="" :BEEP :COLOR 2 :PRINT ADR$+" = "+MC$+" -> "; :INPUT MC1$
1160 IF MC1$="" THEN 1070
1170 POKE ADR,VAL("&H"+MC1$)
1180 COLOR 6 :CFLASH 1 :PRINT "チェック・サム ケイサン チュウ"; :CFLASH 0
1190 GOTO 680 : / — チェック・サム ケイサン
1200 /
1210 / — ツキノ ションヒ ト オフリ ノ ハンタイ —
1220 /
1230 BEEP :COLOR 4 :PRINT :PR=0
1240 PRINT "フ・リント シマスカ [ NO=リターン・キー, YES=1 ] "; :I$=INKEY$(1) :PR=VAL(I$)
1250 IF PR=1 THEN GOSUB 2000
1260 /
1270 AD=AD+128
1280 IF AD<=ED THEN MUSIC "05G3+C" :COLOR 7 :CLS :CFLASH 1 :PRINT "ツキノ ニ スミマス! ケイサン チュウ."; :CF
LASH 0 :GOTO 680
1290 /
1300 CONSOLE 22,3 :CLS :INIT :COLOR 3
1310 PRINT "オフリマシタ。"
1320 MUSIC "04+C3BAGFEDC" :COLOR 7
1330 END
1340 /
1350 /
2000 / ■■■ フ・リント アウト クワール・チン ■■■
2010 /
2020 FOR I=0 TO 15
2030 LPRINT " :"+RIGHT$("000"+HEX$(AD+I*8),4)+" = ";
2040 FOR J=0 TO 7
2050 LPRINT RIGHT$("0"+HEX$(M(I,J)),2)+" ";
2060 NEXT
2070 LPRINT " : "+RIGHT$("0"+HEX$(C(I)),2)
2080 NEXT
2090 LPRINT
2100 FOR I=1 TO 37 :LPRINT "—"; :NEXT :LPRINT
2110 LPRINT " ";
2120 FOR J=0 TO 7
2130 LPRINT RIGHT$("0"+HEX$(D(J)),2)+" ";
2140 NEXT
2150 LPRINT " : "+RIGHT$("0"+HEX$(S),2) :LPRINT :LPRINT
2160 RETURN
2170 /
2180 /
2190 /
2200 / ■■■ マシンコード テーラ ■■■
2210 /
3000 DATA 5E,23,56,D5,E1,36,00,23
3010 DATA 7C,FE,FF,38,F8,C9
3020 /
3030 / ■■■

```


付録4 マシン語 DATA ジェネレータ

```

10 /
20 /
30 /
40 /
50 /
60 /
70 /
80 /
90 /
100 /
110 /
120 /
130 /
140 CLEAR &HD000
150 KBUF ON
160 INIT :CLS 4 :WIDTH 40 :COLOR 4
170 /
180 /
190 /
200 PRINT "マシンの DATA ジェネレータ"
210 PRINT "by Yasuhiro Shimizu"
220 PRINT "1983.10.15"
230 LOCATE 0,5 :COLOR 6
240 PRINT "Address を ニュウリョク スルト、DATA ファン ヲ シットウ"
250 PRINT "セイセイ シマス。"
260 /
270 /
280 /
290 LOCATE 0,10 :COLOR 3
300 INPUT "マシンの START ADDRESS (16シン 4ケタ) = ";ST$:ST=VAL("&H"+ST$) :IF ST<0 THEN ST=ST+65536!
310 IF ST<53248! THEN PRINT CHR$(7,&H1E,5); :GOTO 300
320 COLOR 2
330 INPUT "マシンの END ADDRESS (16シン 4ケタ) = ";ED$:ED=VAL("&H"+ED$) :IF ED<0 THEN ED=ED+65536!
340 IF ED<ST THEN PRINT CHR$(7,&H1E,5); :GOTO 330
350 LOCATE 0,15 :COLOR 7
360 PRINT "タイセイ シマスカ ? [ Y or N ] ";
370 I$=INKEY$(1)
380 IF INSTR(1,"Yン",I$)>0 THEN 140 ELSE IF INSTR(1,"Nん",I$)>0 THEN 400 ELSE 370
390 /
400 /
410 /
420 AD=ST :GYO=30000
430 /
440 IF (AD+7)>ED THEN 590
450 MC$=""
460 FOR I=0 TO 7
470 IF I<7 THEN 480 ELSE 490
480 MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)+", " :GOTO 500
490 MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)
500 AD=AD+1
510 NEXT I
520 D$="DATA "+MC$
530 FLAG=0 :GOTO "シットウセイセイ"
540 GYO=GYO+10
550 GOTO 440
560 /
570 /
580 /
590 MC$=""
600 L=ED-AD :IF L<0 THEN 700
610 FOR I=0 TO L
620 IF I=L THEN 640
630 MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)+", " :GOTO 650
640 MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)
650 AD=AD+1
660 NEXT I
670 D$="DATA "+MC$
680 FLAG=1 :GOTO "シットウセイセイ"
690 /
700 CONSOLE 0,20 :CLS :CONSOLE :COLOR 7 :MUSIC "05G2+C"
710 PRINT "DATA ファン カン セイ シマシタ。"
720 PRINT "DELETE -900 ヲ シツコウシテ"
730 PRINT "ホンタイ・プログラム ヲ ケシテクダサイ !"
740 PRINT
750 END
760 /
770 /
780 /
790 LABEL "シットウセイセイ"
800 /
810 /

```

```
820
830  CONSOLE 22,3
840  LOCATE 0,24 :COLOR 6
850  PRINT CHR$(5);
860  PRINT GY0;D$:PRINT
870  LOCATE 0,23 :COLOR 0
880  KEY0,CHR$(30,30,13)+"GOTO890"+CHR$(13) :END
890  IF FLAG=0 THEN 540
900  GOTO 690
```

付録5 TRON GAME マシン語ソースリスト (「ミニアセンブラ」使用)

Page 00001

```

00010 0000                                #3
00021 0000                                ;
00022 0000                                ;
00023 0000                                ;
00024 0000                                ; M.L. Subs of TRON GAME
00025 0000                                ;
00026 0000                                ; by Y. Shimizu 1983.10.15
00027 0000                                ;
00028 0000                                ;
00029 0000                                ;
00030 0000                                ;
00040 D000                                ORG  $D000
00050 D000                                ;
00060 D000                                ; ■ working area ■
00070 D000                                ;
00080 D000                                ; X coord of TRON
00090 D000                                ; Y coord of TRON
00100 D000                                ; X coord of SARK
00110 D000                                ; Y coord of SARK
00120 D000                                ; move code of TRON
00130 D000                                ; move code of SARK
00140 D000                                ; random
00150 D000                                ; cont flag (0=cont,1=end)
00160 D000                                ; address work (2 bytes)
00170 D000                                ; new address work (2 bytes)
00180 D000                                ; move code work
00190 D000                                ; TRON SARK flag (0=TRON)
00200 D000                                ; character of orbit
00210 D000                                ; character of bike
00220 D000                                ;
00230 D000                                ; ■ Sub of SARK move ■
00240 D000 3E01                                ; SARK flag
00250 D002 320DE0                                ;
00260 D005 CDECD0                                ;
00270 D008 3A06E0                                ;
00280 D00B FE04                                ;
00290 D00D DA44D0                                ;
00300 D010                                ;
00310 D010 3A01E0                                ;
00320 D013 2103E0                                ;
00330 D016 BE                                ; (TRY) < (SKY) ?
00340 D017 DC1BD1                                ;
00350 D01A DAF7D1                                ;
00360 D01D                                ;
00370 D01D 3A01E0                                ;
00380 D020 2103E0                                ;
00390 D023 BE                                ; (TRY) > (SKY) ?
00400 D024 C458D1                                ;
00410 D027 DAF7D1                                ;
00420 D02A                                ;

                                LD A,1
                                (TS),A
                                CALL ADCAL
                                LD A,(RND)
                                CP 4
                                JP C,SELF

                                LD A,(TRY)
                                HL,SKY
                                CP (HL)
                                CALL C,G0UP
                                JP C,CHRPR

                                LD A,(TRY)
                                HL,SKY
                                CP (HL)
                                CALL NZ,G0DN
                                JP C,CHRPR

```

| | | | | |
|-------------------|---------|------|-----------|---------------------------|
| 00430 D02A 3A00E0 | | LD | A, (TRX) | |
| 00440 D02D 2102E0 | | LD | HL, SKX | |
| 00450 D030 BE | | CP | (HL) | ; (TRX) < (SKX) ? |
| 00460 D031 DC91D1 | | CALL | C, GOLT | |
| 00470 D034 DAF7D1 | | JP | C, CHRPR | |
| 00480 D037 | | | | ; |
| 00490 D037 3A00E0 | | LD | A, (TRX) | |
| 00500 D03A 2102E0 | | LD | HL, SKX | |
| 00510 D03D BE | | CP | (HL) | ; (TRX) > (SKX) ? |
| 00520 D03E C4C4D1 | | CALL | NZ, GORT | |
| 00530 D041 DAF7D1 | | JP | C, CHRPR | |
| 00540 D044 | | | | ; |
| 00550 D044 CD1BD1 | SELF: | CALL | GOUP | |
| 00560 D047 DAF7D1 | | JP | C, CHRPR | |
| 00570 D04A CD58D1 | | CALL | GODN | |
| 00580 D04D DAF7D1 | | JP | C, CHRPR | |
| 00590 D050 CD91D1 | | CALL | GOLT | |
| 00600 D053 DAF7D1 | | JP | C, CHRPR | |
| 00610 D056 CDC4D1 | | CALL | GORT | |
| 00620 D059 DAF7D1 | | JP | C, CHRPR | |
| 00630 D05C | | | | ; |
| 00640 D05C | | | | ; Crash of SARK |
| 00650 D05C | | | | ; |
| 00660 D05C 3A0CE0 | | LD | A, (MOVE) | |
| 00670 D05F FE08 | | CP | 8 | ; |
| 00680 D061 2005 | | JR | NZ, NEXT2 | |
| 00690 D063 CD2ED1 | | CALL | GOUP1 | ; |
| 00700 D066 1815 | | JR | CRHSK | |
| 00710 D068 FE02 | NEXT2: | CP | 2 | ; |
| 00720 D06A 2005 | | JR | NZ, NEXT4 | |
| 00730 D06C CD69D1 | | CALL | GODN1 | ; |
| 00740 D06F 180C | | JR | CRHSK | |
| 00750 D071 FE04 | NEXT4: | CP | 4 | ; |
| 00760 D073 2005 | | JR | NZ, NEXT6 | |
| 00770 D075 CD9FD1 | | CALL | GOLT1 | ; |
| 00780 D078 1803 | | JR | CRHSK | |
| 00790 D07A CDD2D1 | NEXT6: | CALL | GORT1 | ; |
| 00800 D07D 3E01 | CRHSK: | LD | A, 1 | |
| 00810 D07F 3207E0 | | LD | (FLAG), A | ; crash flag set for SARK |
| 00820 D082 C3F7D1 | | JP | CHRPR | |
| 00830 D085 | | | | ; |
| 00840 D085 | | | | ; ■ Sub of TRON move ■ |
| 00850 D085 | | | | ; |
| 00860 D085 3E00 | TRONMV: | LD | A, 0 | ; TRON flag |
| 00870 D087 320DE0 | | LD | (TS), A | |
| 00880 D08A CDECD0 | | CALL | ADCAL | |
| 00890 D08D | | | | ; |
| 00900 D08D 3A04E0 | | LD | A, (TRMV) | |
| 00910 D090 210CE0 | | LD | HL, MOVE | |
| 00920 D093 86 | | ADD | A, (HL) | |

| | | | | |
|-------------------|---------|------|-------------|-------------------------|
| 00930 D094 FE0A | | CP | 10 | |
| 00940 D096 2003 | | JR | NZ, KYSCAN | |
| 00950 D098 | | | | ; |
| 00960 D098 2104E0 | GODR: | | LD HL, TRMV | ; Go direct |
| 00970 D09B | | | | ; |
| 00980 D09B 7E | KYSCAN: | | LD A, (HL) | ; Keyscan |
| 00990 D09C FE08 | KEY8: | | CP 8 | |
| 01000 D09E 2010 | | JR | NZ, KEY2 | |
| 01010 D0A0 CD1BD1 | | CALL | GOUP | |
| 01020 D0A3 3808 | | JR | C, EXIT8 | |
| 01030 D0A5 CD2ED1 | | CALL | GOUP1 | |
| 01040 D0A8 3E01 | | LD | A, 1 | |
| 01050 D0AA 3207E0 | | LD | (FLAG), A | |
| 01060 D0AD C3F7D1 | EXIT8: | | JP CHRPR | |
| 01070 D0B0 | | | | ; |
| 01080 D0B0 FE02 | KEY2: | | CP 2 | |
| 01090 D0B2 2010 | | JR | NZ, KEY4 | |
| 01100 D0B4 CD58D1 | | CALL | GODN | |
| 01110 D0B7 3808 | | JR | C, EXIT2 | |
| 01120 D0B9 CD69D1 | | CALL | GODN1 | |
| 01130 D0BC 3E01 | | LD | A, 1 | |
| 01140 D0BE 3207E0 | | LD | (FLAG), A | |
| 01150 D0C1 C3F7D1 | EXIT2: | | JP CHRPR | |
| 01160 D0C4 | | | | ; |
| 01170 D0C4 FE04 | KEY4: | | CP 4 | |
| 01180 D0C6 2010 | | JR | NZ, KEY6 | |
| 01190 D0C8 CD91D1 | | CALL | GOLT | |
| 01200 D0CB 3808 | | JR | C, EXIT4 | |
| 01210 D0CD CD9FD1 | | CALL | GOLT1 | |
| 01220 D0D0 3E01 | | LD | A, 1 | |
| 01230 D0D2 3207E0 | | LD | (FLAG), A | |
| 01240 D0D5 C3F7D1 | EXIT4: | | JP CHRPR | |
| 01250 D0D8 | | | | ; |
| 01260 D0D8 FE06 | KEY6: | | CP 6 | |
| 01270 D0DA 20BC | | JR | NZ, GODR | |
| 01280 D0DC CDC4D1 | | CALL | GORT | |
| 01290 D0DF 3808 | | JR | C, EXIT6 | |
| 01300 D0E1 CDD2D1 | | CALL | GORT1 | |
| 01310 D0E4 3E01 | | LD | A, 1 | |
| 01320 D0E6 3207E0 | | LD | (FLAG), A | |
| 01330 D0E9 C3F7D1 | EXIT6: | | JP CHRPR | |
| 01340 D0EC | | | | ; |
| 01350 D0EC | | | | ; ■ Subs of movement ■ |
| 01360 D0EC | | | | ; |
| 01370 D0EC 3A0DE0 | ADCAL: | | LD A, (TS) | ; Address calculation ■ |
| 01380 D0EF FE00 | | CP | 0 | |
| 01390 D0F1 200C | | JR | NZ, SKPART | |
| 01400 D0F3 | | | | ; |
| 01410 D0F3 7E | TRPART: | | LD A, (HL) | ; (HL) = Ten key |
| 01420 D0F4 320CE0 | | LD | (MOVE), A | |

| | | | |
|-------------------|---------|---------------|-------------------------|
| 01430 D0F7 2100E0 | LD | HL, TRX | |
| 01440 D0FA 56 | LD | D, (HL) | ; D ← (TRX) |
| 01450 D0FB 23 | INC | HL | |
| 01460 D0FC 5E | LD | E, (HL) | ; E ← (TRY) |
| 01470 D0FD 180C | JR | XFER | |
| 01480 D0FF | | | ; |
| 01490 D0FF 3A05E0 | SKPART: | LD A, (SKMV) | |
| 01500 D102 320CE0 | LD | (MOVE), A | |
| 01510 D105 2102E0 | LD | HL, SKX | |
| 01520 D108 56 | LD | D, (HL) | ; D ← (SKX) |
| 01530 D109 23 | INC | HL | |
| 01540 D10A 5E | LD | E, (HL) | ; E ← (SKY) |
| 01550 D10B | | | ; |
| 01560 D10B 210030 | XFER: | LD HL, \$3000 | ; HL ← VRAM Top address |
| 01570 D10E 4A | LD | C, D | |
| 01590 D10F 1600 | LD | D, 0 | |
| 01600 D111 0628 | LD | B, 40 | |
| 01610 D113 19 | LOOP1: | ADD HL, DE | |
| 01620 D114 10FD | DJNZ | LOOP1 | |
| 01630 D116 09 | ADD | HL, BC | ; HL ← HL+E*40+D |
| 01640 D117 2208E0 | LD | (ADR), HL | |
| 01650 D11A C9 | RET | | |
| 01660 D11B | | | ; |
| 01670 D11B 2A08E0 | GOUP: | LD HL, (ADR) | ; Go up ? _____ |
| 01680 D11E 112800 | LD | DE, 40 | |
| 01690 D121 B7 | OR | A | |
| 01700 D122 ED52 | SBC | HL, DE | ; HL ← HL-40 |
| 01710 D124 44 | LD | B, H | |
| 01720 D125 4D | LD | C, L | |
| 01730 D126 ED78 | IN | A, (C) | |
| 01740 D128 FE20 | CP | \$20 | ; up ? |
| 01750 D12A 280B | JR | Z, GOUP2 | |
| 01760 D12C B7 | OR | A | ; Cy ← 0 |
| 01770 D12D C9 | RET | | |
| 01780 D12E | | | ; |
| 01790 D12E 2A08E0 | GOUP1: | LD HL, (ADR) | ; Go up ! _____ |
| 01800 D131 112800 | LD | DE, 40 | |
| 01810 D134 B7 | OR | A | |
| 01820 D135 ED52 | SBC | HL, DE | |
| 01830 D137 220AE0 | GOUP2: | LD (NADR), HL | |
| 01840 D13A 3E08 | LD | A, 8 | ; up code = 8 |
| 01850 D13C 320CE0 | LD | (MOVE), A | |
| 01860 D13F 3A0DE0 | LD | A, (TS) | |
| 01870 D142 FE00 | CP | 0 | |
| 01880 D144 2009 | JR | NZ, SKUP | |
| 01890 D146 | | | ; |
| 01900 D146 3A01E0 | TRUP: | LD A, (TRY) | ; Go up for TRON |
| 01910 D149 3D | DEC | A | |
| 01920 D14A 3201E0 | LD | (TRY), A | ; (TRY) ← (TRY)-1 |
| 01930 D14D 1807 | JR | UPFLG | |

```

01940 D14F
01950 D14F 3A03E0      SKUP:          LD A, (SKY)          ;
01960 D152 3D          DEC A          ; Go up for SARK
01970 D153 3203E0      LD (SKY), A      ; (SKY) <- (SKY)-1
01980 D156 37          UPFLG:        SCF
01990 D157 C9          RET
02000 D158
02010 D158 2A08E0      GODN:          LD HL, (ADR)          ;
02020 D15B 112800      LD DE, 40      ; Go down ? _____
02030 D15E 19          ADD HL, DE      ; HL <- HL+40
02040 D15F 44          LD B, H
02050 D160 4D          LD C, L
02060 D161 ED78        IN A, (C)
02070 D163 FE20        CP #20          ; down ?
02080 D165 2809        JR Z, GODN2
02090 D167 B7          OR A          ; Cy <- 0
02100 D168 C9          RET
02110 D169
02120 D169 2A08E0      GODN1:         LD HL, (ADR)          ;
02130 D16C 112800      LD DE, 40      ; Go down ! _____
02140 D16F 19          ADD HL, DE
02150 D170 220AE0      GODN2:         LD (NADR), HL
02160 D173 3E02        LD A, 2          ; down code = 2
02170 D175 320CE0      LD (MOVE), A
02180 D178 3A0DE0      LD A, (TS)
02190 D17B FE00        CP 0
02200 D17D 2009        JR NZ, SKDN
02210 D17F
02220 D17F 3A01E0      TRDN:          LD A, (TRY)          ;
02230 D182 3C          INC A          ;
02240 D183 3201E0      LD (TRY), A      ; (TRY) <- (TRY)+1
02250 D186 1807        JR DNFLG
02260 D188
02270 D188 3A03E0      SKDN:          LD A, (SKY)          ;
02280 D18B 3C          INC A          ;
02290 D18C 3203E0      LD (SKY), A      ; (SKY) <- (SKY)+1
02300 D18F 37          DNFLG:        SCF
02310 D190 C9          RET
02320 D191
02330 D191 2A08E0      GOLT:          LD HL, (ADR)          ;
02340 D194 2B          DEC HL          ; Go left ? _____
02350 D195 44          LD B, H          ; HL <- HL-1
02360 D196 4D          LD C, L
02370 D197 ED78        IN A, (C)
02380 D199 FE20        CP #20          ; left ?
02390 D19B 2806        JR Z, GOLT2
02400 D19D B7          OR A          ; Cy <- 0
02410 D19E C9          RET
02420 D19F
02430 D19F 2A08E0      GOLT1:         LD HL, (ADR)          ;
; Go left ! _____

```

```

02440 D1A2 2B
02450 D1A3 220AE0
02460 D1A6 3E04
02470 D1A8 320CE0
02480 D1A8 3A0DE0
02490 D1AE FE00
02500 D1B0 2009
02510 D1B2
02520 D1B2 3A00E0
02530 D1B5 3D
02540 D1B6 3200E0
02550 D1B9 1807
02560 D1BB
02570 D1BB 3A02E0
02580 D1BE 3D
02590 D1BF 3202E0
02600 D1C2 37
02610 D1C3 C9
02620 D1C4
02630 D1C4 2A08E0
02640 D1C7 23
02650 D1C8 44
02660 D1C9 4D
02670 D1CA ED78
02680 D1CC FE20
02690 D1CE 2806
02700 D1D0 B7
02710 D1D1 C9
02720 D1D2
02730 D1D2 2A08E0
02740 D1D5 23
02750 D1D6 220AE0
02760 D1D9 3E06
02770 D1DB 320CE0
02780 D1DE 3A0DE0
02790 D1E1 FE00
02800 D1E3 2009
02810 D1E5
02820 D1E5 3A00E0
02830 D1E8 3C
02840 D1E9 3200E0
02850 D1EC 1807
02860 D1EE
02870 D1EE 3A02E0
02880 D1F1 3C
02890 D1F2 3202E0
02900 D1F5 37
02910 D1F6 C9
02920 D1F7
02930 D1F7

GOLT2:      DEC HL
              LD (NADR),HL
              LD A,4          ; left code = 4
              LD (MOVE),A
              LD A,(TS)
              CP 0
              JR NZ,SKLT
              ;
TRLT:        LD A,(TRX)
              DEC A
              LD (TRX),A      ; (TRX) <- (TRX)-1
              JR LTFLG
              ;
SKLT:        LD A,(SKX)
              DEC A
              LD (SKX),A      ; (SKX) <- (SKX)-1
              SCF
LTFLG:       RET
              ;
GORT:        LD HL,(ADR)      ; Go right ?
              INC HL          ; HL <- HL+1
              LD B,H
              LD C,L
              IN A,(C)
              CP $20          ; right ?
              JR Z,GORT2
              OR A             ; Cy <- 0
              RET
              ;
GORT1:       LD HL,(ADR)      ; Go right !
              INC HL
GORT2:       LD (NADR),HL
              LD A,6          ; right code = 6
              LD (MOVE),A
              LD A,(TS)
              CP 0
              JR NZ,SKRT
              ;
TRRT:        LD A,(TRX)
              INC A
              LD (TRX),A      ; (TRX) <- (TRX)+1
              JR RTFLG
              ;
SKRT:        LD A,(SKX)
              INC A
              LD (SKX),A      ; (SKX) <- (SKX)+1
              SCF
RTFLG:       RET
              ;
              ; Character print

```



```

02940 D1F7
02950 D1F7 CD26D2      CHRPR:      CALL CHRDET      ;
02960 D1FA
02970 D1FA ED4B08E0      LD      BC, (ADR)      ;
02980 D1FE 1600      LD      D, 0      ; orbit character select
02990 D200 3A0EE0      LD      A, (ORBIT)
03000 D203 5F      LD      E, A
03010 D204 CD3CD3      CALL PRINT
03020 D207
03030 D207 3A07E0      LD      A, (FLAG)
03040 D20A FE00      CP      0
03050 D20C 200D      JR      NZ, CRHCHR
03060 D20E
03070 D20E ED4B0AE0      BIKCHR:      LD BC, (NADR)
03080 D212 1601      LD      D, 1      ; bike character select
03090 D214 3A0FE0      LD      A, (BIKE)
03100 D217 5F      LD      E, A
03110 D218 C33CD3      JP      PRINT
03120 D21B
03130 D21B ED4B0AE0      CRHCHR:      LD BC, (NADR)
03140 D21F 1602      LD      D, 2      ; crash character select
03150 D221 1EE8      LD      E, $E8      ; E ← "X"
03160 D223 C33CD3      JP      PRINT
03170 D226
03180 D226
03190 D226
03200 D226 3A0CE0      CHRDET:      LD A, (MOVE)
03210 D229
03220 D229 FE02      MVSCAN:      CP 2
03221 D22B CA79D2      JP      Z, CHRDN
03222 D22E FE04      CP      4
03223 D230 CABAD2      JP      Z, CHRLT
03224 D233 FE06      CP      6
03225 D235 CAFBD2      JP      Z, CHRRT
03230 D238
03240 D238 3A0DE0      CHRUP:      LD A, (TS)
03250 D238 FE00      CP      0
03260 D23D 2005      JR      NZ, SKUPOB
03270 D23F 3A04E0      TRUPOB:      LD A, (TRMV)
03280 D242 1803      JR      LTUP
03290 D244 3A05E0      SKUPOB:      LD A, (SKMV)
03300 D247 FE04      LTUP:      CP 4
03310 D249 2004      JR      NZ, RTUP
03320 D24B 3E99      LD      A, $99      ; A ← "L"
03330 D24D 180A      JR      UPOB
03340 D24F FE06      RTUP:      CP 6
03350 D251 2004      JR      NZ, UPUP
03360 D253 3E98      LD      A, $98      ; A ← "J"
03370 D255 1802      JR      UPOB
03380 D257 3E91      UPUP:      LD A, $91      ; A ← "I"

```

```

03390 D259 320EE0      UPOB:          LD (ORBIT),A
03400 D25C              ;
03410 D25C 3A0DE0      LD A,(TS)
03420 D25F FE00        CP 0
03430 D261 200B        JR NZ,SKUPBK
03440 D263 3E64      TRUPBK:      LD A,100      ; TRON up bike
03450 D265 320FE0      LD (BIKE),A
03460 D268 3E08        LD A,8
03470 D26A 3204E0      LD (TRMV),A      ; (TRMV) <- 8
03480 D26D C9          RET
03490 D26E 3EC8      SKUPBK:      LD A,200      ; SARK up bike
03500 D270 320FE0      LD (BIKE),A
03510 D273 3E08        LD A,8
03520 D275 3205E0      LD (SKMV),A      ; (SKMV) <- 8
03530 D278 C9          RET
03540 D279              ;
03570 D279 3A0DE0      CHRDN:      LD A,(TS)      ; character down
03580 D27C FE00        CP 0
03590 D27E 2005        JR NZ,SKDN0B
03600 D280 3A04E0      TRDN0B:      LD A,(TRMV)
03610 D283 1803        JR LTDN
03620 D285 3A05E0      SKDN0B:      LD A,(SKMV)
03630 D288 FE04      LTDN:          CP 4
03640 D28A 2004        JR NZ,RTDN
03650 D28C 3E9A        LD A,$9A      ; A <- "r"
03660 D28E 180A        JR DN0B
03670 D290 FE06      RTDN:          CP 6
03680 D292 2004        JR NZ,DNDN
03690 D294 3E97        LD A,$97      ; A <- "j"
03700 D296 1802        JR DN0B
03710 D298 3E91      DNDN:          LD A,$91      ; A <- "i"
03720 D29A 320EE0      DN0B:          LD (ORBIT),A
03730 D29D              ;
03740 D29D 3A0DE0      LD A,(TS)
03750 D2A0 FE00        CP 0
03760 D2A2 200B        JR NZ,SKDNBK
03770 D2A4 3E6E      TRDNBK:      LD A,110      ; TRON down bike
03780 D2A6 320FE0      LD (BIKE),A
03790 D2A9 3E02        LD A,2
03800 D2AB 3204E0      LD (TRMV),A      ; (TRMV) <- 2
03810 D2AE C9          RET
03820 D2AF 3ED2      SKDNBK:      LD A,210      ; SARK down bike
03830 D2B1 320FE0      LD (BIKE),A
03840 D2B4 3E02        LD A,2
03850 D2B6 3205E0      LD (SKMV),A      ; (SKMV) <- 2
03860 D2B9 C9          RET
03870 D2BA              ;
03900 D2BA 3A0DE0      CHRLT:      LD A,(TS)      ; character left
03910 D2BD FE00        CP 0
03920 D2BF 2005        JR NZ,SKLT0B

```

```

03930 D2C1 3A04E0      TRLT0B:      LD A, (TRMV)
03940 D2C4 1803          JR          UPLT
03950 D2C6 3A05E0      SKLT0B:      LD A, (SKMV)
03960 D2C9 FE08          UPLT:      CP 8
03970 D2CB 2004          JR          NZ, DNLT
03980 D2CD 3E97          LD          A, $97          ; A <- "1"
03990 D2CF 180A          JR          LTOB
04000 D2D1 FE02          DNLT:      CP 2
04010 D2D3 2004          JR          NZ, LTLT
04020 D2D5 3E98          LD          A, $98          ; A <- "1"
04030 D2D7 1802          JR          LTOB
04040 D2D9 3E90          LTLT:      LD A, $90          ; A <- "-"
04050 D2DB 320EE0      LTOB:      LD (ORBIT), A
04060 D2DE              ;
04070 D2DE 3A0DE0          LD          A, (TS)
04080 D2E1 FE00          CP          0
04090 D2E3 200B          JR          NZ, SKLTBK
04100 D2E5 3E78          TRLTBK:      LD A, 120          ; TRON left bike
04110 D2E7 320FE0          LD          (BIKE), A
04120 D2EA 3E04          LD          A, 4
04130 D2EC 3204E0          LD          (TRMV), A          ; (TRMV) <- 4
04140 D2EF C9            RET
04150 D2F0 3EDC          SKLTBK:      LD A, 220          ; SARK left bike
04160 D2F2 320FE0          LD          (BIKE), A
04170 D2F5 3E04          LD          A, 4
04180 D2F7 3205E0          LD          (SKMV), A          ; (SKMV) <- 4
04190 D2FA C9            RET
04200 D2FB              ;
04210 D2FB 3A0DE0      CHRRT:      LD A, (TS)          ; character right
04220 D2FE FE00          CP          0
04230 D300 2005          JR          NZ, SKRT0B
04240 D302 3A04E0      TRRT0B:      LD A, (TRMV)
04250 D305 1803          JR          UPRT
04260 D307 3A05E0      SKRT0B:      LD A, (SKMV)
04270 D30A FE08          UPRT:      CP 8
04280 D30C 2004          JR          NZ, DNRT
04290 D30E 3E9A          LD          A, $9A          ; A <- "r"
04300 D310 180A          JR          RTOB
04310 D312 FE02          DNRT:      CP 2
04320 D314 2004          JR          NZ, RTRT
04330 D316 3E99          LD          A, $99          ; A <- "L"
04340 D318 1802          JR          RTOB
04350 D31A 3E90          RTRT:      LD A, $90          ; A <- "-"
04360 D31C 320EE0      RTOB:      LD (ORBIT), A
04370 D31F              ;
04380 D31F 3A0DE0          LD          A, (TS)
04390 D322 FE00          CP          0
04400 D324 200B          JR          NZ, SKRTBK
04410 D326 3E82          TRRTBK:      LD A, 130          ; TRON right bike
04420 D328 320FE0          LD          (BIKE), A

```

```

04430 D32B 3E06          LD      A,6
04440 D32D 3204E0        LD      (TRMV),A          ; (TRMV) <- 6
04450 D330 C9             RET
04460 D331 3EE6           SKRTBK: LD      A,230          ; SARK right bike
04470 D333 320FE0        LD      (BIKE),A
04480 D336 3E06           LD      A,6
04490 D338 3205E0        LD      (SKMV),A          ; (SKMV) <- 6
04500 D33B C9             RET
04510 D33C               ;
04515 D33C               ;
04520 D33C 7A            PRINT: LD      A,D          ; print sub █████
04530 D33D FE01          CP      1
04540 D33F 2004          JR      NZ,OBATR
04550 D341 3E27           SKATR: LD      A,$27          ; COLOR 7, CGEN 1 (bike)
04560 D343 181F          JR      ATROUT
04570 D345 3A0DE0        OBATR: LD      A,(TS)
04580 D348 FE00          CP      0
04590 D34A 200D          JR      NZ,SKATR
04600 D34C 7A            TRATR: LD      A,D
04610 D34D FE00          CP      0
04620 D34F 2004          JR      NZ,TRCRSH
04630 D351 3E26           LD      A,$26          ; COLOR 6, CGEN 1 (TR orbit)
04640 D353 180F          JR      ATROUT
04650 D355 3E06           TRCRSH: LD      A,$06          ; COLOR 6, CGEN 0 (TR X)
04660 D357 180B          JR      ATROUT
04670 D359 7A            SKATR: LD      A,D
04680 D35A FE00          CP      0
04690 D35C 2004          JR      NZ,SKCRSH
04700 D35E 3E22           LD      A,$22          ; COLOR 2, CGEN 1 (SK orbit)
04710 D360 1802          JR      ATROUT
04720 D362 3E02           SKCRSH: LD      A,$02          ; COLOR 2, CGEN 0 (SK X)
04730 D364               ;
04740 D364 CBA0          ATROUT: RES      4,B          ; attribute address
04750 D366 ED79          OUT      (C),A          ; attribute out
04760 D368 CBE0          CHROUT: SET      4,B          ; text VRAM address
04770 D36A 7B            LD      A,E          ; A <- character code
04780 D36B ED79          OUT      (C),A          ; character out
04790 D36D C9             RET
04800 D36E               ;
04810 D36E               ; █████ Character reverse █████
04820 D36E               ;
04830 D36E 010020        CREV: LD      BC,$2000          ; attribute top address
04840 D371 ED78          LPREV: IN      A,(C)
04850 D373 CB0F          SET      3,A          ; CREV flag set
04860 D375 ED79          OUT      (C),A          ; attribute out
04870 D377 03            INC      BC
04880 D378 78            LD      A,B
04890 D379 FE23          CP      $23
04900 D37B 38F4          JR      C,LPREV
04910 D37D 79            LD      A,C

```

Page 00011

04920 D37E FEE8
04930 D380 38EF
04940 D382 C9
04950 D383
60000 D383

CP \$E8 ; < 23E8H ?
JR C,LPREV
RET ;
END

付録6 X1, X1C, X1D
 マニュアル対応表

| 本文ページ | X1 マニュアルページ | X1C マニュアルページ | X1D マニュアルページ |
|-------|----------------|-----------------|-----------------|
| 2 | 175 | 200 | 224 |
| 2 | 195 | 220 | ※ |
| 4 | 107 | 112 | 120 |
| 5 | 175～177 | 200～202 | 224～226 |
| 38 | 176 | 201 | 225 |
| 42 | 61 | 62 | 64 |
| 74 | 91 | 94 | 102 |
| 76 | 72 | 72 | 80 |
| 81 | 168 | 192～ | 216～ |
| 82 | 62 | 63 | 65 |
| 117 | 176 | 201 | 225 |
| 125 | 176 | 201 | 225 |
| 125 | 183 | 209 | 236 |
| 127 | 41 | 42 | 44 |
| 127 | 20 | 20 | 22 |
| 144 | 178 | 203～204 | 227～229 |
| 146 | 178 | 203～204 | 227～229 |
| 152 | 183 | 209 | 236 |
| 201 | 34 | 34 | 36 |
| 211 | 154 | 162 | 170 |

※ X1DのマニュアルにはBASICシステムプログラムの掲載されて
 おりません。

《さくいん》

〔ア行〕

| | |
|------------------|-------|
| アキュムレータ | 25 |
| アクセス | 84 |
| アスキーコード | 16・77 |
| アスキーセーブ | 200 |
| アスキーダンプ | 16 |
| アセンブラ | 23 |
| アセンブリ言語 | 23 |
| アセンブル | 23 |
| アトリビュート | 79 |
| アトリビュートVRAM ... | 80 |
| アドレス | 9 |
| インタプリタ | 41 |
| オペランド | 67 |
| オペレーションコード | 67 |

〔カ行〕

| | |
|-----------------|-------|
| 間接アドレス指定 | 60 |
| キャリーフラグ | 94・97 |
| 高級言語 | 41 |
| 高水準言語 | 41 |
| コマンドレベル | 5 |
| コントロールコード | 78 |
| コンパイラ | 41 |

〔サ行〕

| | |
|----------------|-------|
| サブルーチン | 117 |
| システムサブルーチン ... | 133 |
| 16進表記法 | 10 |
| 上下位逆転の原則 | 31・58 |
| 条件ジャンプ命令 | 98 |
| スタック | 119 |
| スタックポインタ | 119 |

| | |
|----------------|-------|
| 絶対アドレス指定 | 108 |
| ゼロフラグ | 94・97 |
| 専用レジスタ | 48 |
| 相対アドレス指定 | 108 |
| 相対ジャンプ命令 | 109 |
| ソース | 26 |

〔タ行〕

| | |
|-----------------|-----|
| ダンプ | 15 |
| チェックサム | 193 |
| 直接アドレス指定 | 60 |
| ディスプレイメント | 59 |
| デスティネーション | 26 |
| デバッグ | 36 |
| 転送命令 | 26 |

〔ナ行〕

| | |
|--------------|----|
| ニーモニック | 22 |
| 入出力装置 | 13 |
| 入出力ポート | 83 |

〔ハ行〕

| | |
|----------------|-------|
| バイト | 8 |
| ハンドアセンブル | 23 |
| 汎用レジスタ | 48 |
| 比較命令 | 95 |
| ビット | 8 |
| ビット操作命令 | 91 |
| ビットパターン | 20 |
| 符号付数 | 112 |
| プッシュ | 120 |
| フラグ | 93 |
| フラグレジスタ | 48・93 |
| フリップフロップ | 46 |

| | |
|-----------------|-----|
| プログラムカウンタ | 110 |
| 補 数 | 111 |
| 補助レジスタ | 48 |
| ポップ | 121 |

〔マ行〕

| | |
|------------------|-------|
| マイクロコンピュータ | 8 |
| マージ | 201 |
| マシンコード | 23 |
| マシン語 | 20・23 |
| メモリー | 9 |
| メモリーマップ | 125 |
| モニター | 4 |

〔ラ行〕

| | |
|---------------|-------|
| リロケータブル | 107 |
| リンク | 144 |
| レジスタ | 25・46 |

〔ワ行〕

| | |
|--------------|-----|
| ワークエリア | 166 |
|--------------|-----|

| | |
|----------------|-------|
| A レジスタ | 25 |
| CPIJ | 7 |
| Cy フラグ | 94・97 |
| FAC | 152 |
| F レジスタ | 48・93 |
| I/O | 13 |
| I/O アドレス | 83 |
| IPL | 42 |
| LSB | 9 |
| LSI | 8 |
| MSB | 9 |
| PC レジスタ | 110 |

| | |
|-------------|-------|
| RAM | 42 |
| ROM | 42 |
| SP | 119 |
| VRAM | 75 |
| Z フラグ | 94・97 |

著者紹介

清水 保弘 (しみず やすひろ)

1954年、横浜市生れ。東京大学理学部数学科卒業。現在、東京都立大学大学院博士課程に在学中。専攻は数学（幾何学、群論）。数学研究におけるパソコン利用の可能性について追究している

誰にでもわかる

△▽ マシン語入門

著者 清水 保弘

定価 2,800円

1983 © Y. Shimizu

昭和59年3月1日 第2刷発行

印刷・製本

昭和工芸印刷(株)

(234)1801



株 日本ソフト&ハード社

出版部 ☎03(209)2585



コンピュータ・イラン

- | | | | |
|-------------|------|----------------------------|------------------|
| 本 社 | 〒160 | 東京都豊島区高田3-11-14 藤間ビル | ☎03-232-0541(代) |
| 関 西 支 社 | 〒530 | 大阪市北区堂島2-2-2 近鉄堂島ビル7F | ☎06-341-7621(代) |
| 名 古 屋 支 社 | 〒453 | 名古屋市中村区椿町1-16 リクルート名古屋ビル5F | ☎052-451-7371(代) |
| 東京高田馬場店 | 〒160 | 東京都新宿区高田馬場2-17-4 菊月ビル3F | ☎03-209-7376(代) |
| 東京新宿西口店 | 〒160 | 東京都新宿区西新宿1-9-13 高倉第2ビル1F | ☎03-342-4821(代) |
| 横 浜 西 口 店 | 〒220 | 横浜市西区南幸2-5-4 深沢ビル1F | ☎045-312-4611(代) |
| 名 古 屋 駅 前 店 | 〒453 | 名古屋市中村区椿町1-16 リクルート名古屋ビル5F | ☎052-451-7371(代) |
| ニ ュ ー 梅 田 店 | 〒530 | 大阪市北区堂島2-2-2 近鉄堂島ビル7F | ☎06-346-1552(代) |
| 神 戸 三 宮 店 | 〒650 | 神戸市中央区三宮町2-1-5 センタープラザ西館3F | ☎078-332-3961 |



[株]日本ソフト&ハード社 出版物案内

X-1プログラムライブラリー (仮 称)

清水 保 弘 著 近日発行

X-1 マシン語 〈中級編〉 (仮 称)

清水 保 弘 著 刊行予定

誰にでもわかる

FM-7/8 マシン語の本

6809CPUの機械をFM-7/8を使って完全マスター。スロットマシン・ワードプロセッサのソースリストを完全解説。

高 橋 嘉 規 著 定価 2,800円

おもしろまじめ F-BASIC

F-BASIC (FM-7) のコマンドを簡易に解説。マニュアルだけではわからないコマンドの使い方を詳しく説明。誰にでもわかるBASIC入門書。

伊 地 知 芳 樹 共著 定価 2,800円
渋谷 好 則

誰にでもわかる

マシン語ゲームのつくり方

6809・Z80の機械語を使ったゲーム作成のノウハウが勉強できる。PC 8001MK II・FM-7・FM-8・日立レベル3の各機種に対応。

高 橋 嘉 規 著 定価 2,800円

誰にでもわかる

パーソナルコンピュータのデータ処理

パソコンのデータファイル管理の実際を完全理解。VISICALC・MULTIPLAN・SUPERCALC等の簡易言語の選び方、使い方。

浅茅原竹毘古 著 定価 2,800円

誰にでもわかる

6809アセンブラ

日立ベーシックマスターレベル3・FM-8を使って6809機械語ゲームの作り方を学ぶ。アセンブリ言語の基本を初心者にわかりやすく解説。

天 野 康 孝 著 定価 3,800円



[株]日本ソフト&ハード社

〒171 東京都豊島区高田3-11-14 藤間ビル2F
☎ (03) 232-0541 出版部 (03) 209-2585



 **[株]日本ソフト&ハード社**

〒171 東京都豊島区高田3-11-14 藤間ビル ☎ (03) 232-0541代

¥2,800